RTXC™

Real-Time Executive in C

Smart Starts Here℠

COURSE CONTENT

- Day 1
  - Unit 1: RTXC Overview
  - Unit 2: Application Development Process
    - Exercises
  - Unit 3: Tasks and Memory Partitions
    - Exercises
- Day 2
  - Unit 4: Semaphores, Timers and Resources
    - Exercises
  - Unit 5: Queues, Mailboxes and Messages
    - Exercises
  - Unit 6: Interrupt Handling

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC TRAINING

UNIT 1

RTXC Overview

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC V3.2 PRODUCT OVERVIEW

- Out-of-the-Box use with your compiler and processor
- Written in ANSI C for portability
- 600+ page RTXC User's Manual
- 7 kernel object classes using 72 Kernel Services
- Event driven, multitasking design with Preemptive scheduling
- Support for Time-Sliced and Round Robin scheduling
- Configurable, ROMable and extensible
- System level debug utility - RTXCbug
- System generation utility - SYSgen

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC IS PROCESSOR INDEPENDENT

- Common API for all processors
- Common RTXC User's Manual
- RTXC for a particular processor is called a PORT

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC IS COMPILER SPECIFIC

- There are differences between C compilers
- RTXC built for a particular C compiler is called a BINDING
- Details of binding to a compiler are found in the Binding Manual

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

USER'S MANUAL CONTENT

- Theory of Operation (Section 2)
- Functional overview - organization and content of RTXC kernel objects (Section 3)
- Tutorial (Section 4)
- RTXC Kernel Services reference (Section 5)
- RTXCgen - system configuration utility (Section 6)
- Device drivers and interrupt service routines (Section 7)
- RTXCbug - system level debugging tool (Section 8)
- Application notes (Section 9)

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

MULTITASKING

CHARACTERISTICS

- Formal structure is provided by policies and rules implemented as Kernel Services
- User's application is organized into a set of tasks
- Application Program Interface (API) provides a standard interface between application tasks and the kernel services

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC POLICIES AND RULES

- POLICIES establish the fundamental principles of the kernel design
- RULES implement these policies

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC POLICIES

FUNDAMENTAL PRINCIPLES

- Allocation of system resources
- Scheduling methods
- Responsiveness and performance
- Flexibility

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC BASIC POLICIES

- RTXC should contain sufficient numbers and types of services to make the kernel useful to a variety of applications
- RTXC should employ a multitasking design to achieve maximum CPU efficiency
- RTXC primary multitasking method should be event driven
- Each task should have a priority indicative of its relative scheduling importance

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC BASIC POLICIES

- RAM usage should be kept to a minimum
- RTXC should impose a minimum overhead on the application
- RTXC performance should be deterministic to the greatest extent possible

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC BASIC RULES

- The Current Task is the highest priority task in the system that is not blocked
- The Current Task maintains control of the CPU either until it runs to completion, yields, becomes blocked by resource unavailability or is preempted
- If a task of higher priority than the Current Task becomes ready, it preempts the lower priority task and becomes the Current Task

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC BASIC RULES

- The RTXC Kernel is interruptible but not reentrant
- Interrupt Service Routines may use only those services expressly intended for ISR usage
- The NULL Task is always the lowest priority task and its priority must never be changed
- The NULL Task must never be blocked

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC
TASK SCHEDULING

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

PREEMPTIVE SCHEDULING

SPECIFIC CHARACTERISTICS

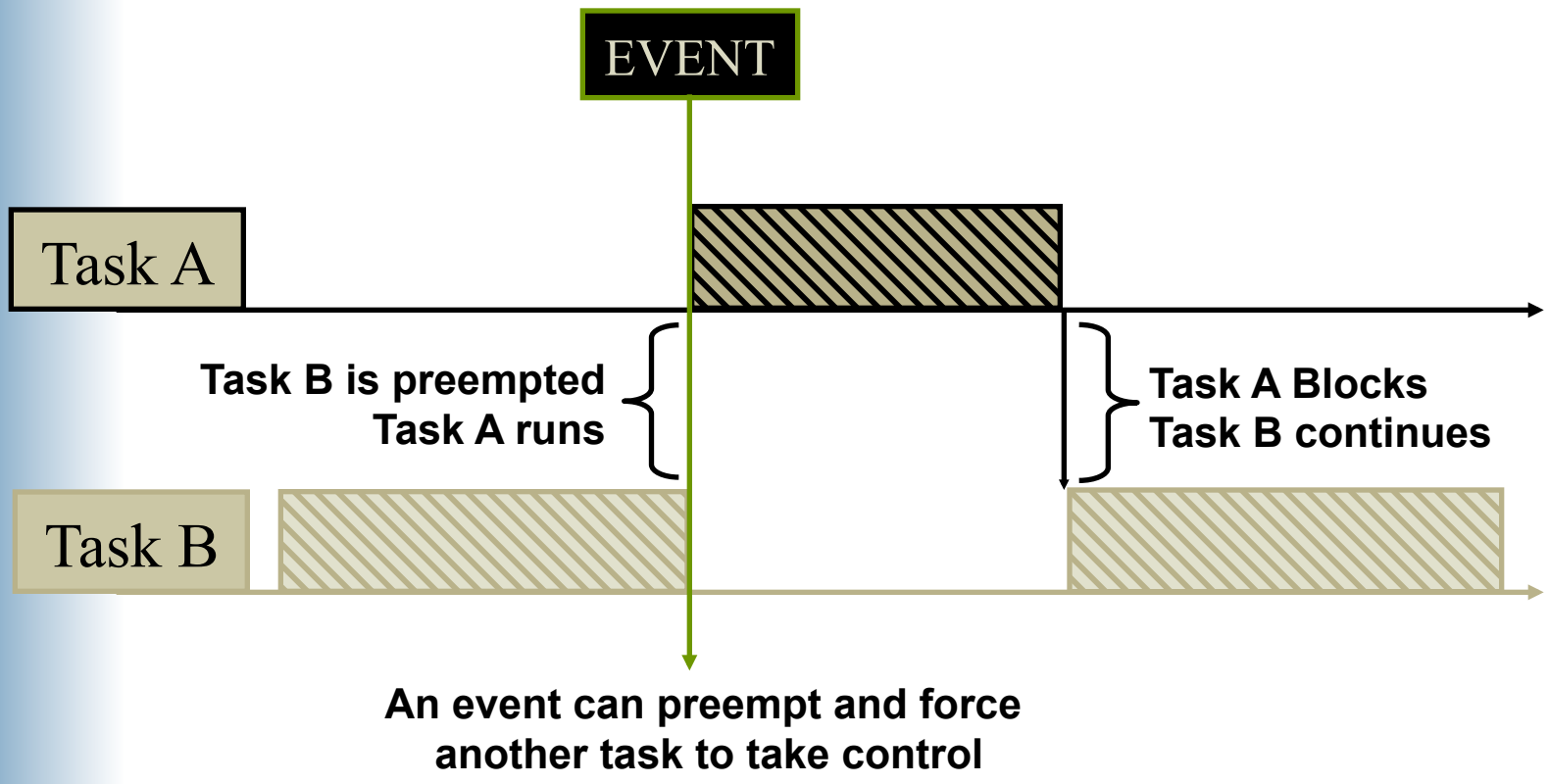
- Tasks have priorities to indicate their relative importance
- Highest priority ready task gets control of processor
- Tasks execute in strict priority sequence with preemptive task scheduling
- Current Task runs until it terminates, is preempted, blocked, or yields control
- A task can control its own runtime by terminating, blocking or yielding
- Preemption is handled by the Scheduler

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

PREEMPTIVE SCHEDULING

REACTING TO AN EVENT



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

PREEMPTIVE SCHEDULING

SUMMARY

- Most important ready task at any given time controls the CPU
- Synchronized with events through the use of task priorities
- A deterministic design is possible because response time to an event is predictable

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

TIME-SLICED SCHEDULING

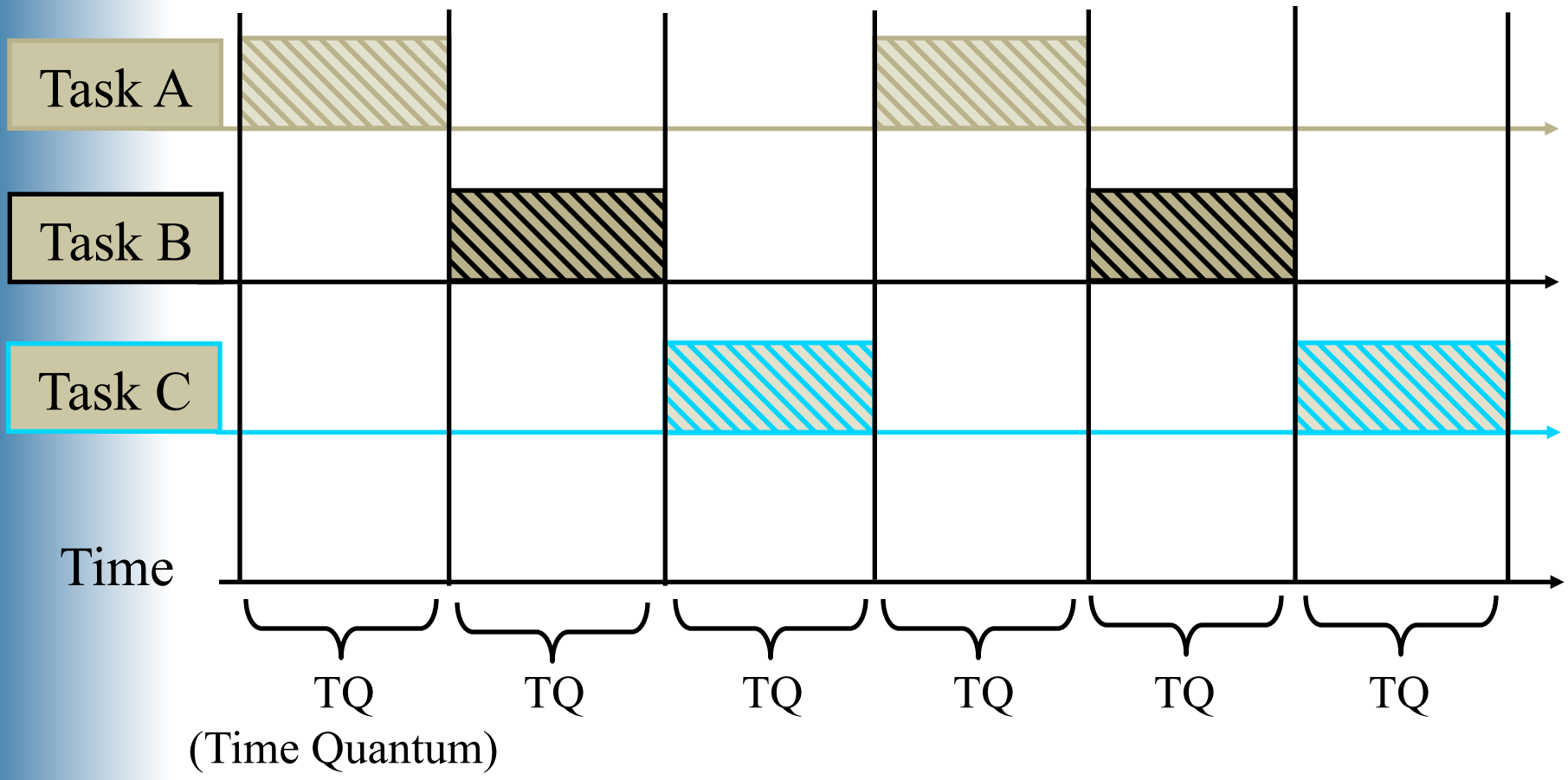
SPECIFIC CHARACTERISTICS

- No task monopolizes CPU because execution time is limited by a time quantum
- Granularity of time quantum determines system responsiveness
- Process must be predictable so that time quantum can be chosen correctly

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

TIME-SLICED SCHEDULING

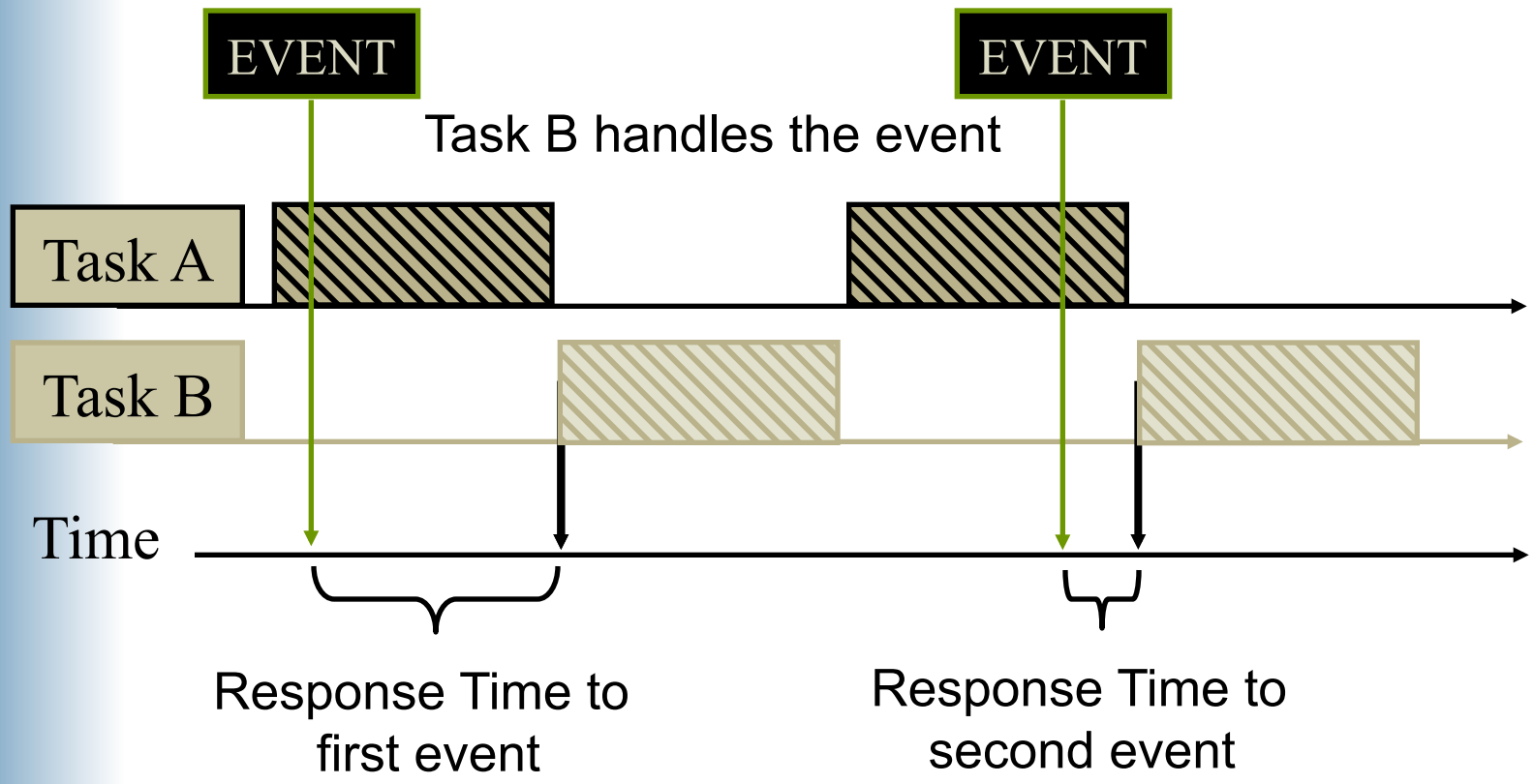


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

TIME-SLICED SCHEDULING

NON-DETERMINISTIC RESPONSE TIME

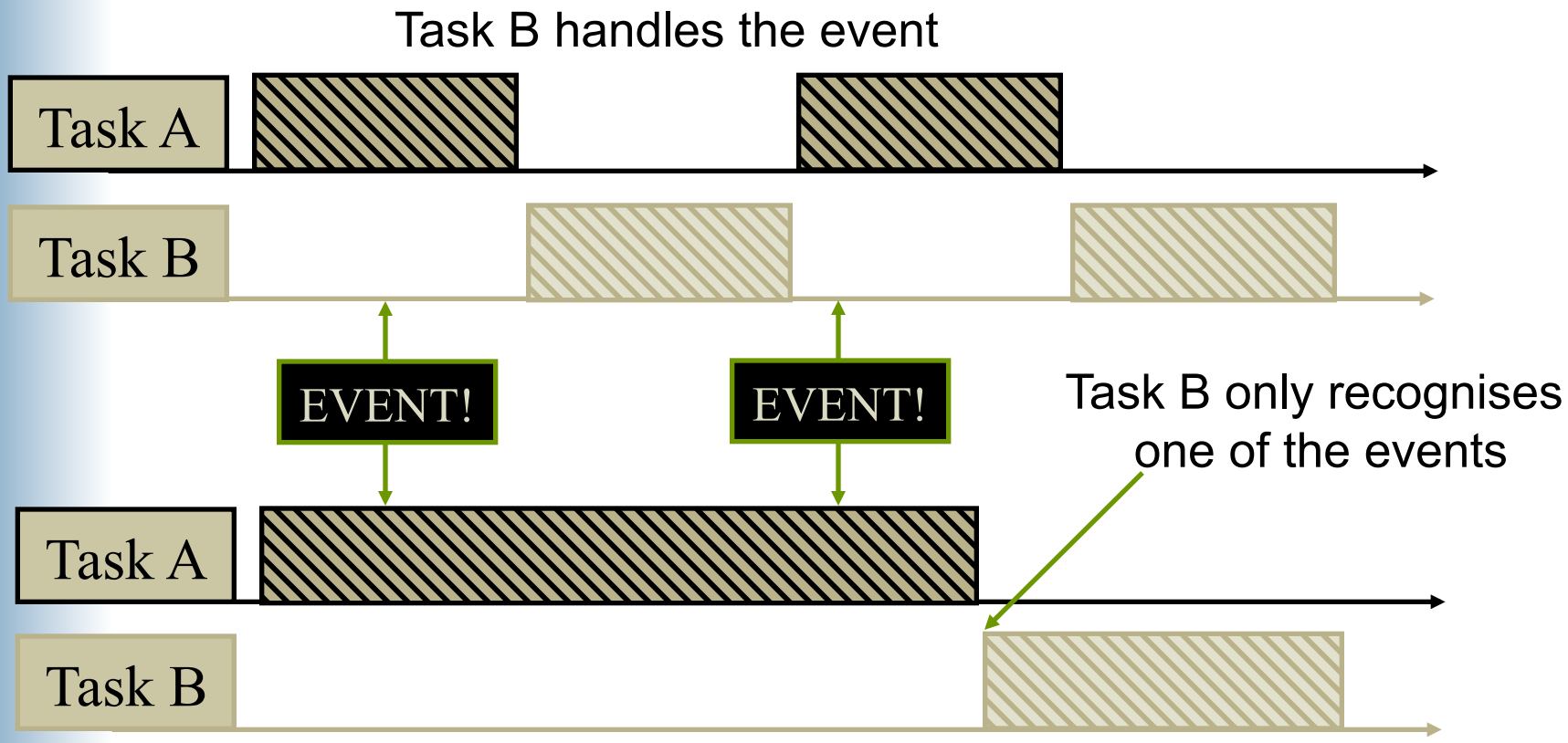


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

TIME-SLICED SCHEDULING

LARGE TIME QUANTUM: MISSING AN EVENT



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

TIME-SLICED SCHEDULING

TIME QUANTUM CAUTION

- There is an overhead associated with each task switch
- If the time quantum is too small then the overhead may become excessive and lead to undesirable system performance
- If the time quantum is too large then critical events may be missed entirely

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

TIME-SLICED SCHEDULING

SUMMARY

- No task monopolizes the CPU 100%
- Non-deterministic response to events
- Adjustable worse case response time obtained by changing the time quantum

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

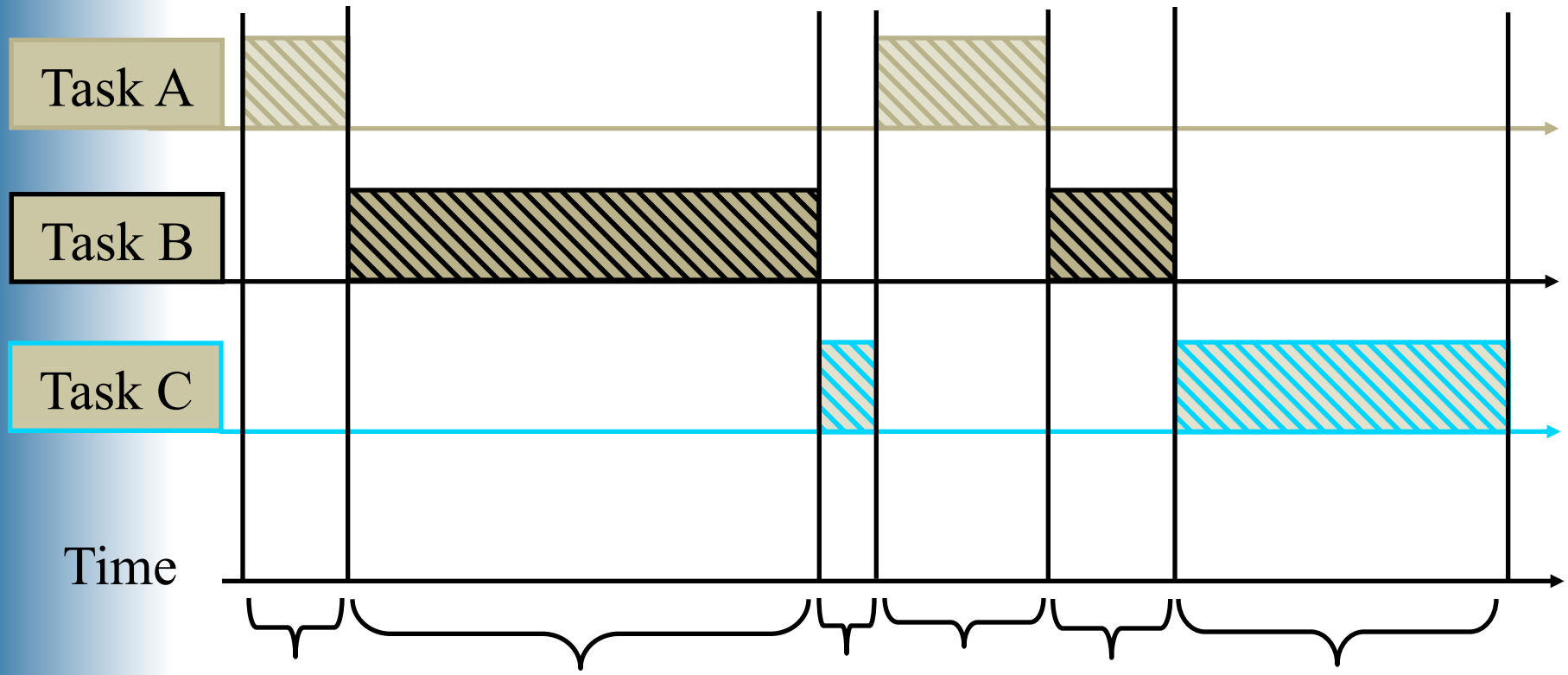
ROUND ROBIN SCHEDULING

- System functions are organized into modules
- Individual task modules execute sequentially, with or without time limit, in one loop

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

ROUND ROBIN SCHEDULING



Each task runs until it terminates, thus we have unequal CPU usage

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

ROUND ROBIN SCHEDULING

SUMMARY

- Application has 100% of CPU
- Modular design
- Unstable timing between successive passes of loop
- No consideration is given to relative importance of modules, making event response time non-predictable

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TRAINING

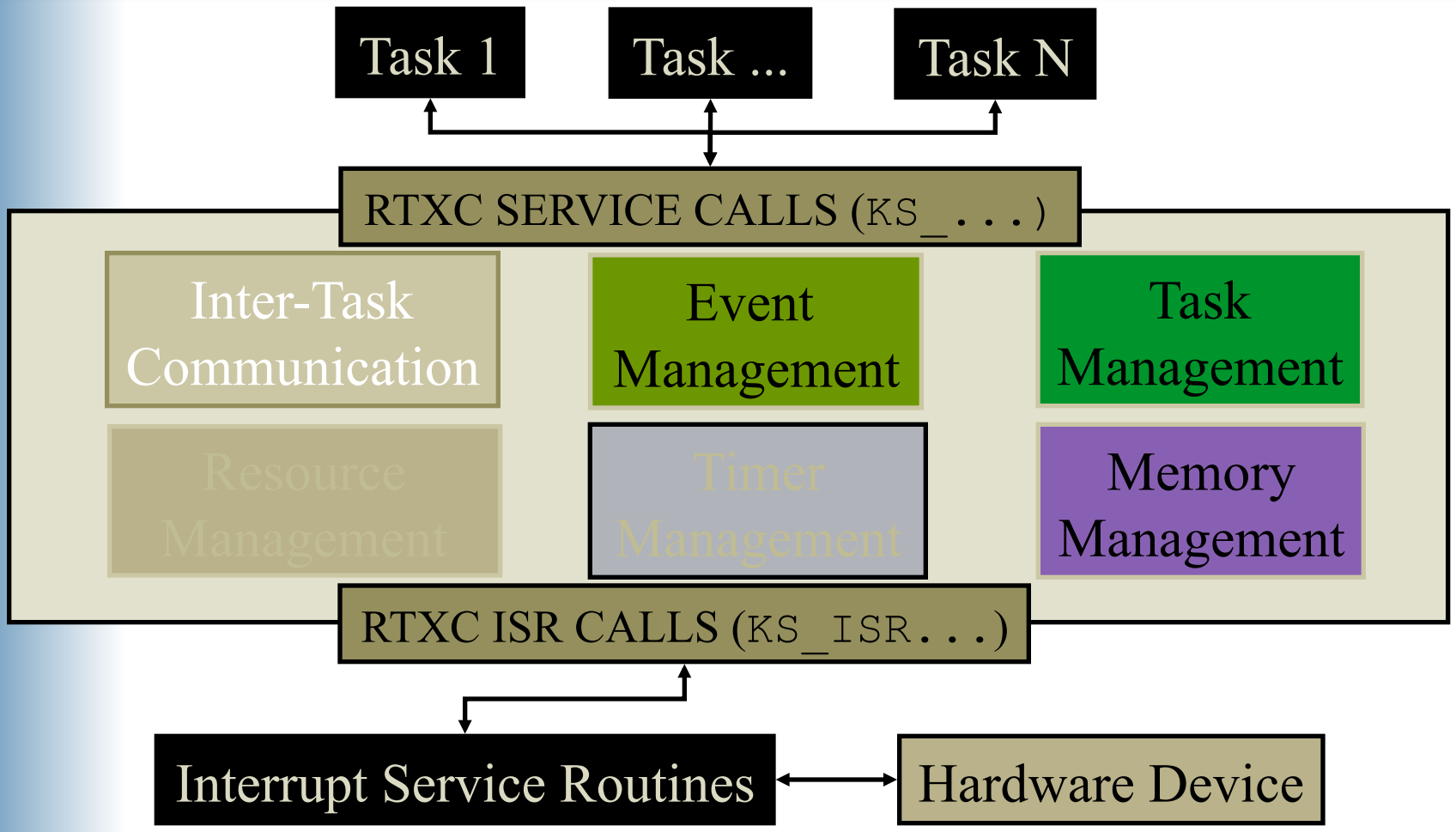
UNIT 2

Application Development Process

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

APPLICATION USING RTXC



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

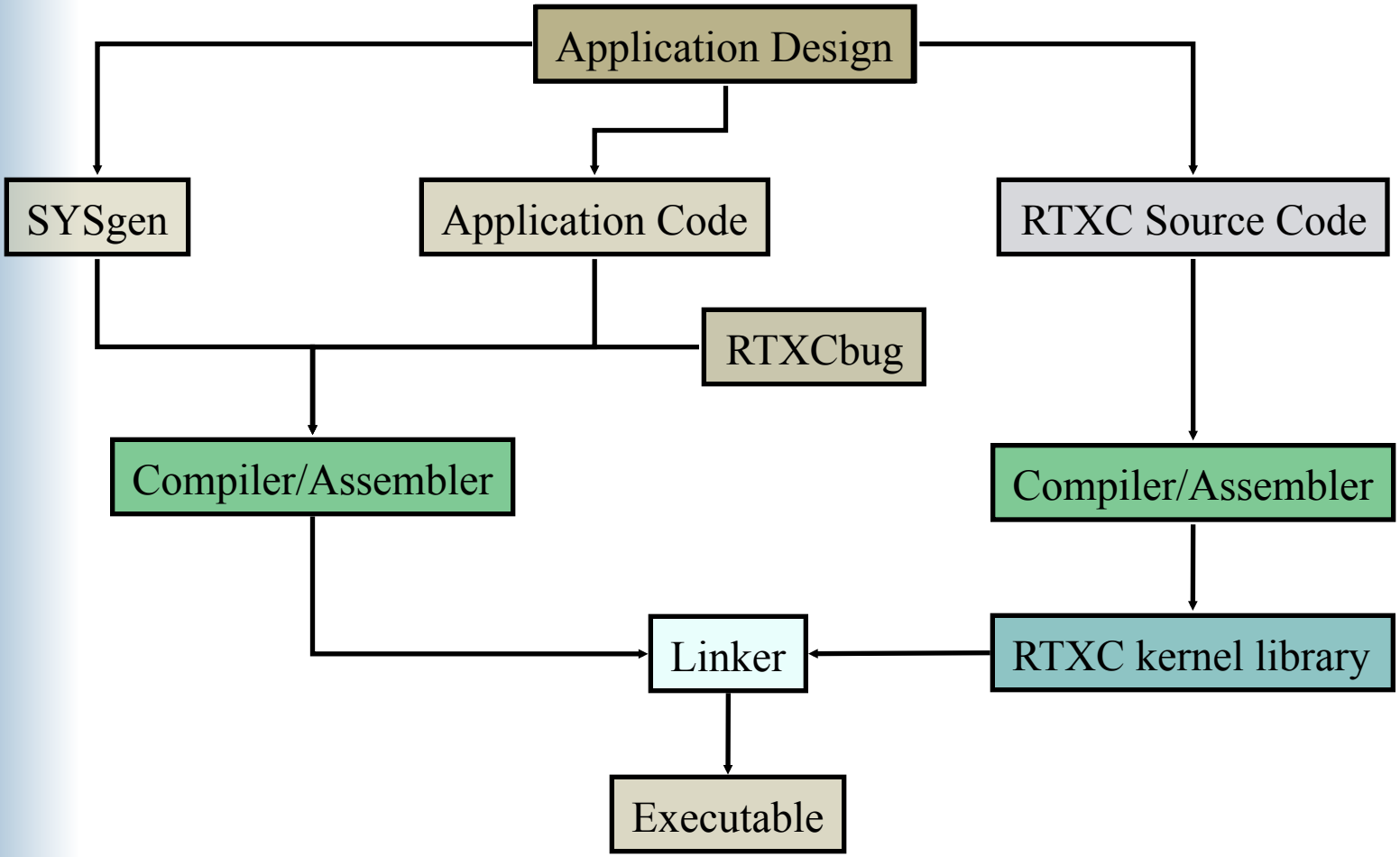
Quadros Systems Inc.

APPLICATION DESIGN

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

APPLICATION DEVELOPMENT PROCESS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

APPLICATION DESIGN SUMMARY

- Define requirements of the system
- Define necessary Kernel Objects
- Define tasks and their functionality
  - **Define priorities and synchronization**
  - **Define communication mechanisms**
- Ensure compliance with rules and policies of RTXC for efficient design
- Iterative process - use information from RTXCbug to refine system design

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

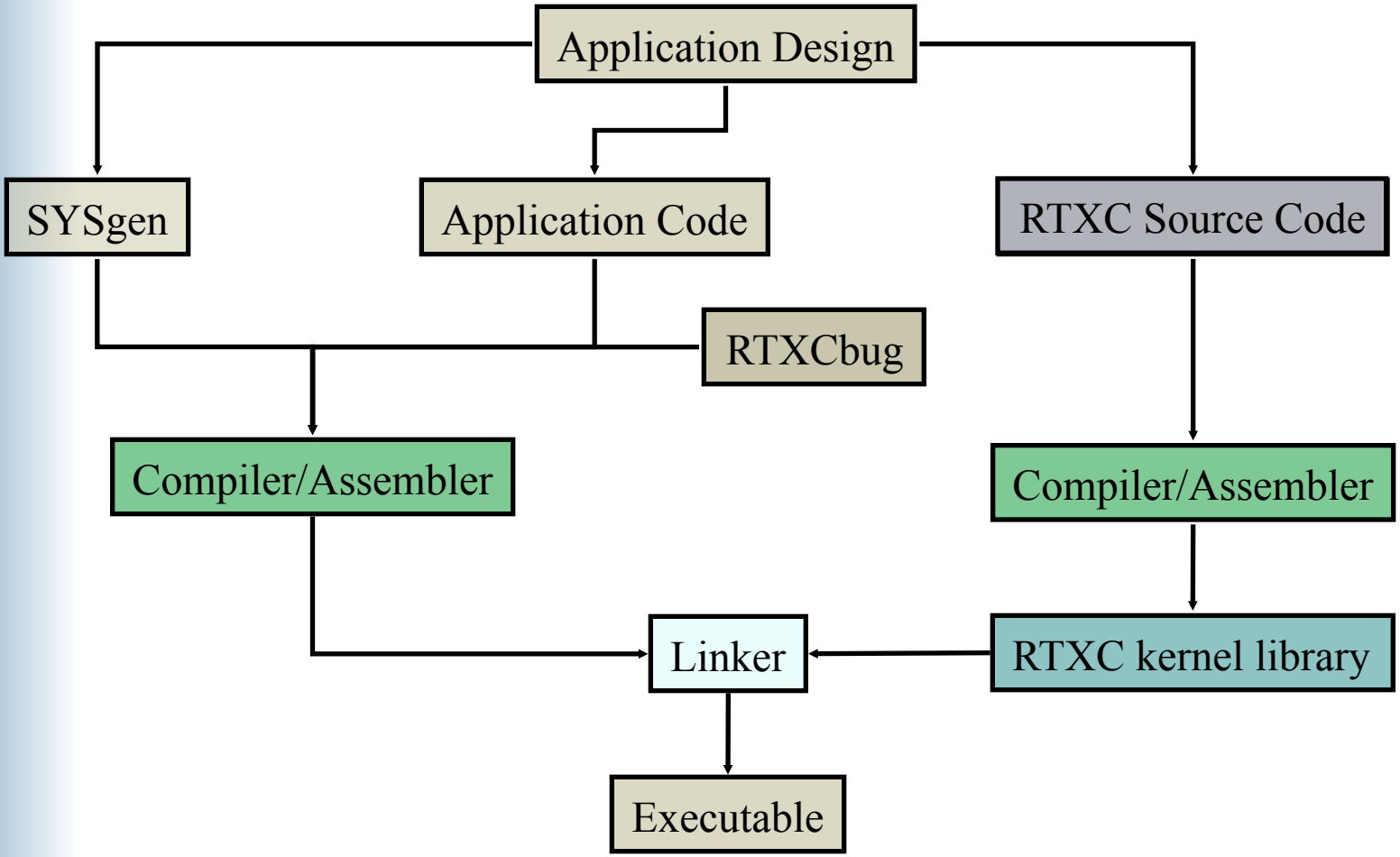
Quadros™ Systems Inc.

RTXC
SOURCE CODE

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

APPLICATION DEVELOPMENT PROCESS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC SOURCE CODE SUMMARY

- Most configuration options are set in rtxcopts.h
- Some options are set in the make file
- Should require few changes once configured
- The source code is compiled and the object files are added to a library - librtxc.lib

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

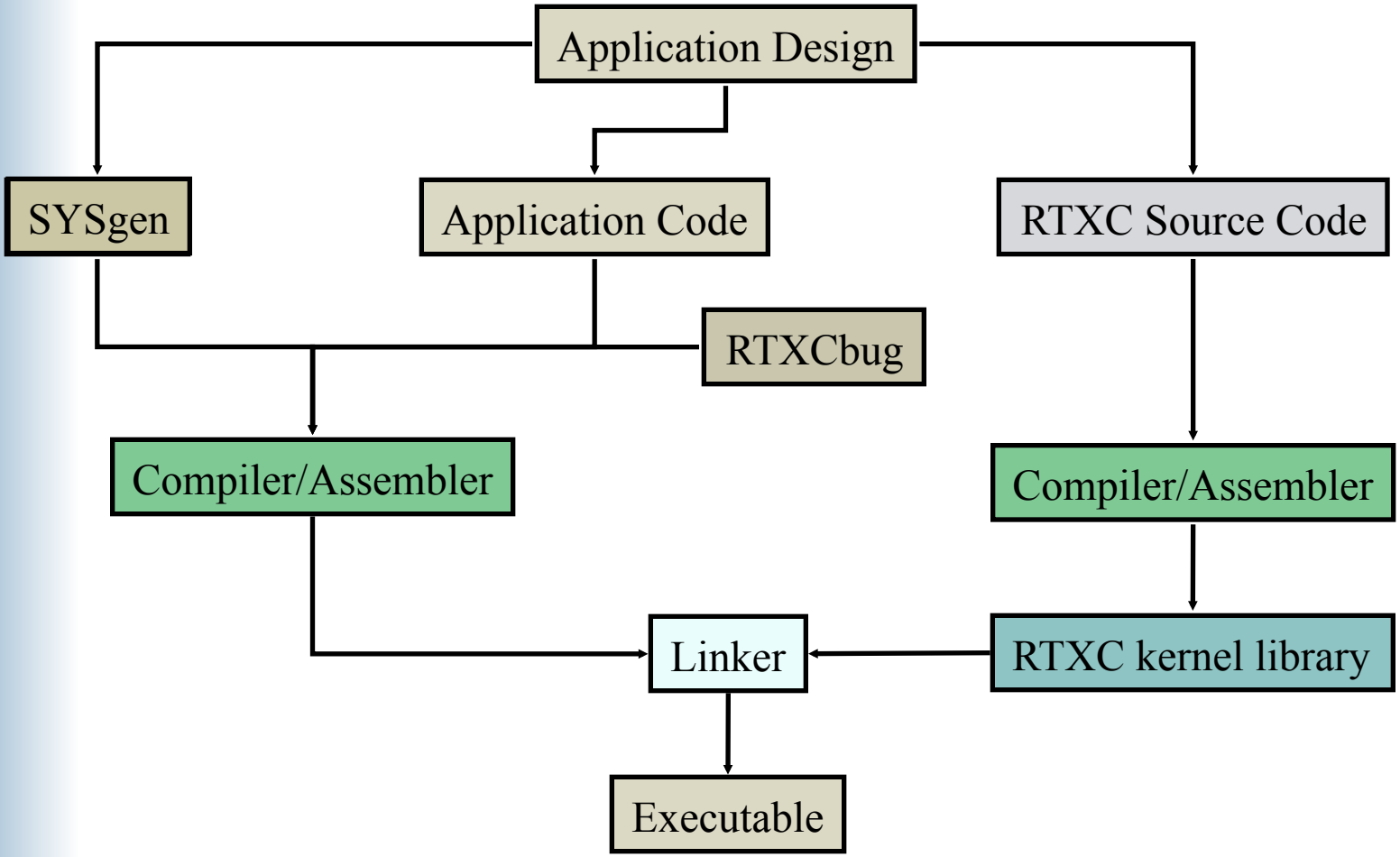
Quadros™ Systems Inc.

SYSgen

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

APPLICATION DEVELOPMENT PROCESS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

INTERACTIVE SYSTEM GENERATION UTILITY

- Predefinition of system configuration
- Produces C data structures for all kernel objects
- Uncomplicated and efficient definitions
- Fast turnaround of configuration changes
- Error free code of system configuration
- Self documenting

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen

- Creates .rtx, .c, and .h files
  - .rtx files used by SYSgen for configuration
  - .c files allocate space and instantiate objects
  - .h files declare constants for the objects

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen

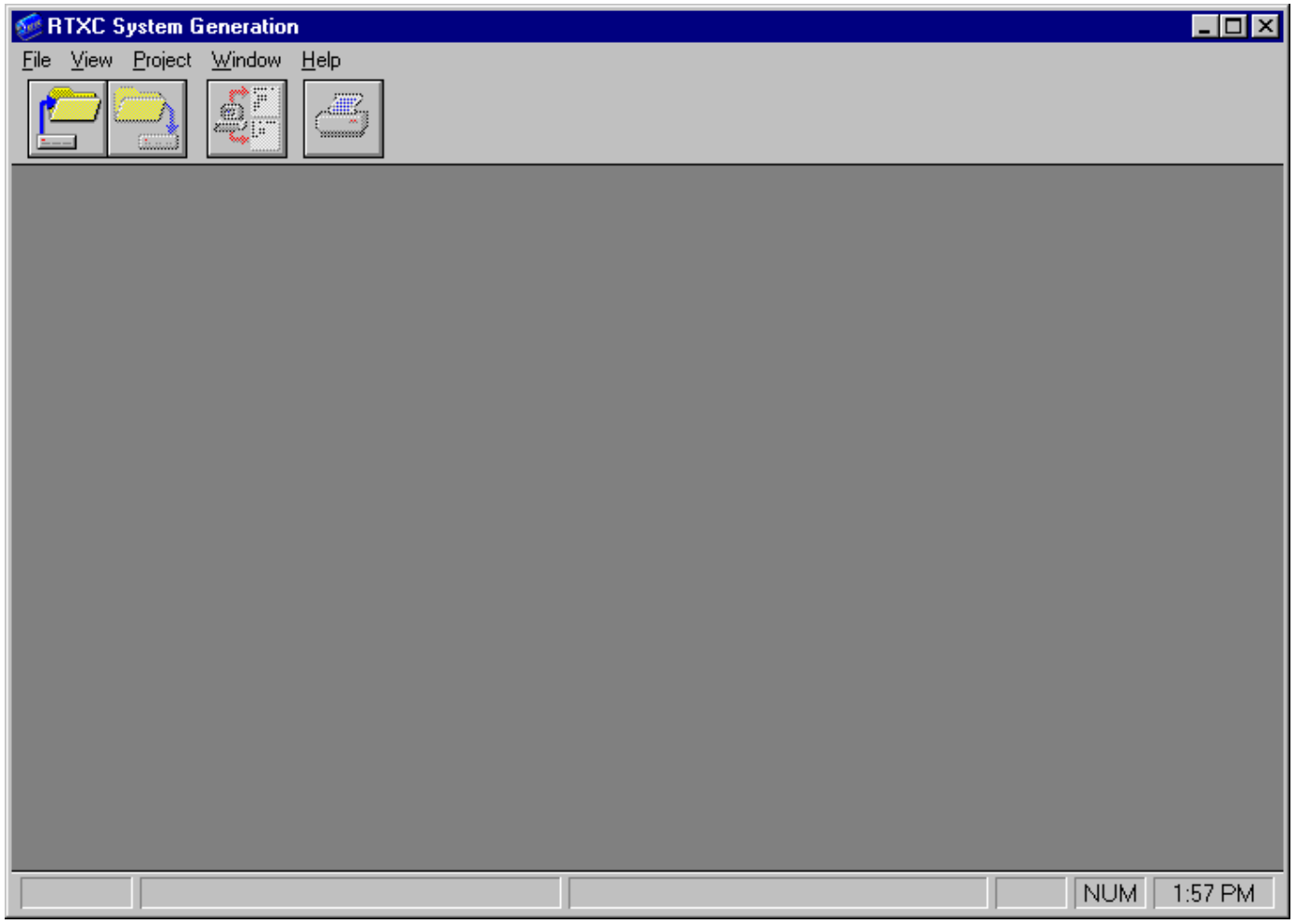
- Kernel Object names integrated with C Header files
- Accessible to:
  - Application code
  - RTXCbug displays and dialog
- Consistent method of editing all Kernel Objects.

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

MAIN MENU

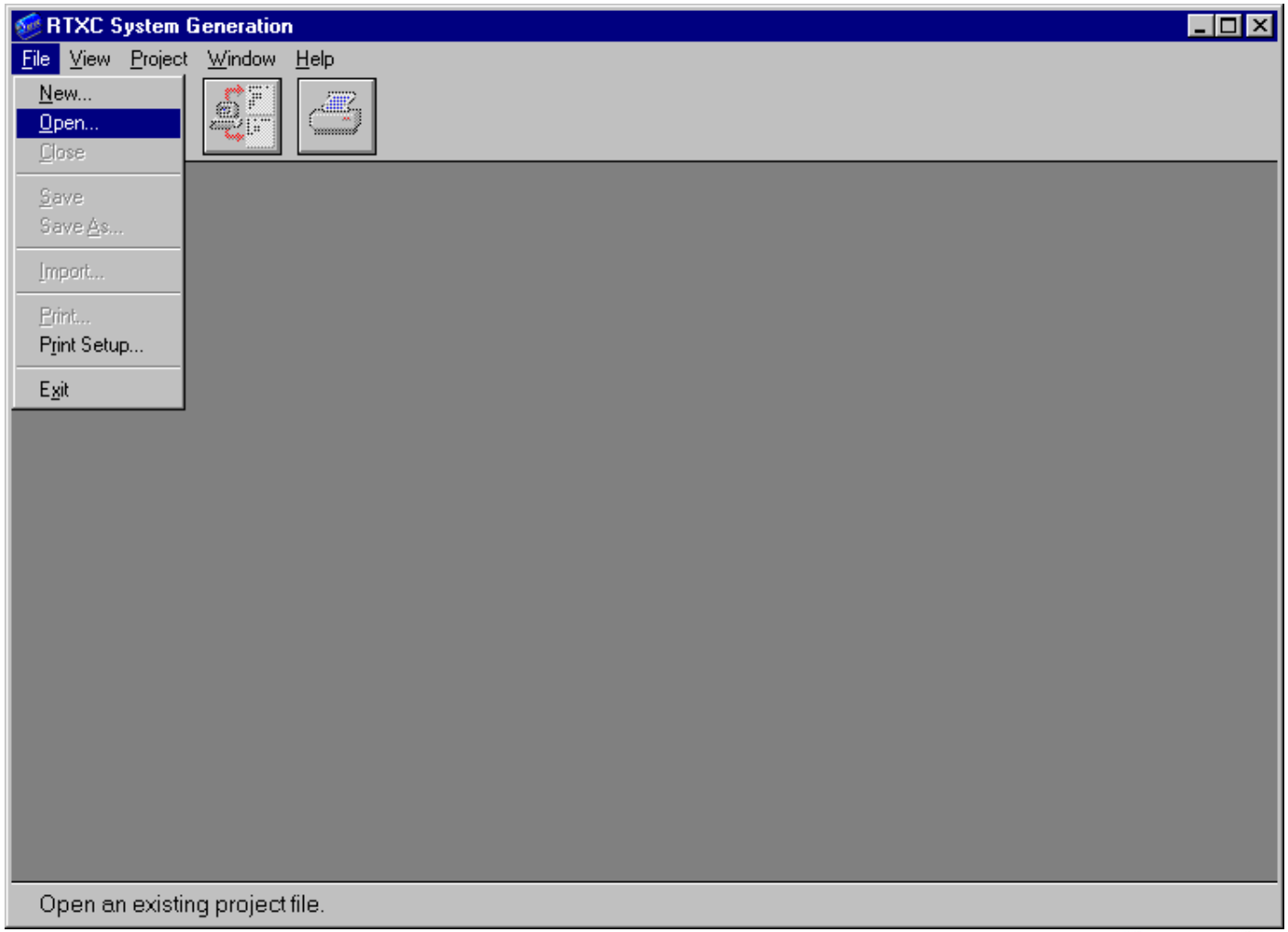


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

OPEN PROJECT

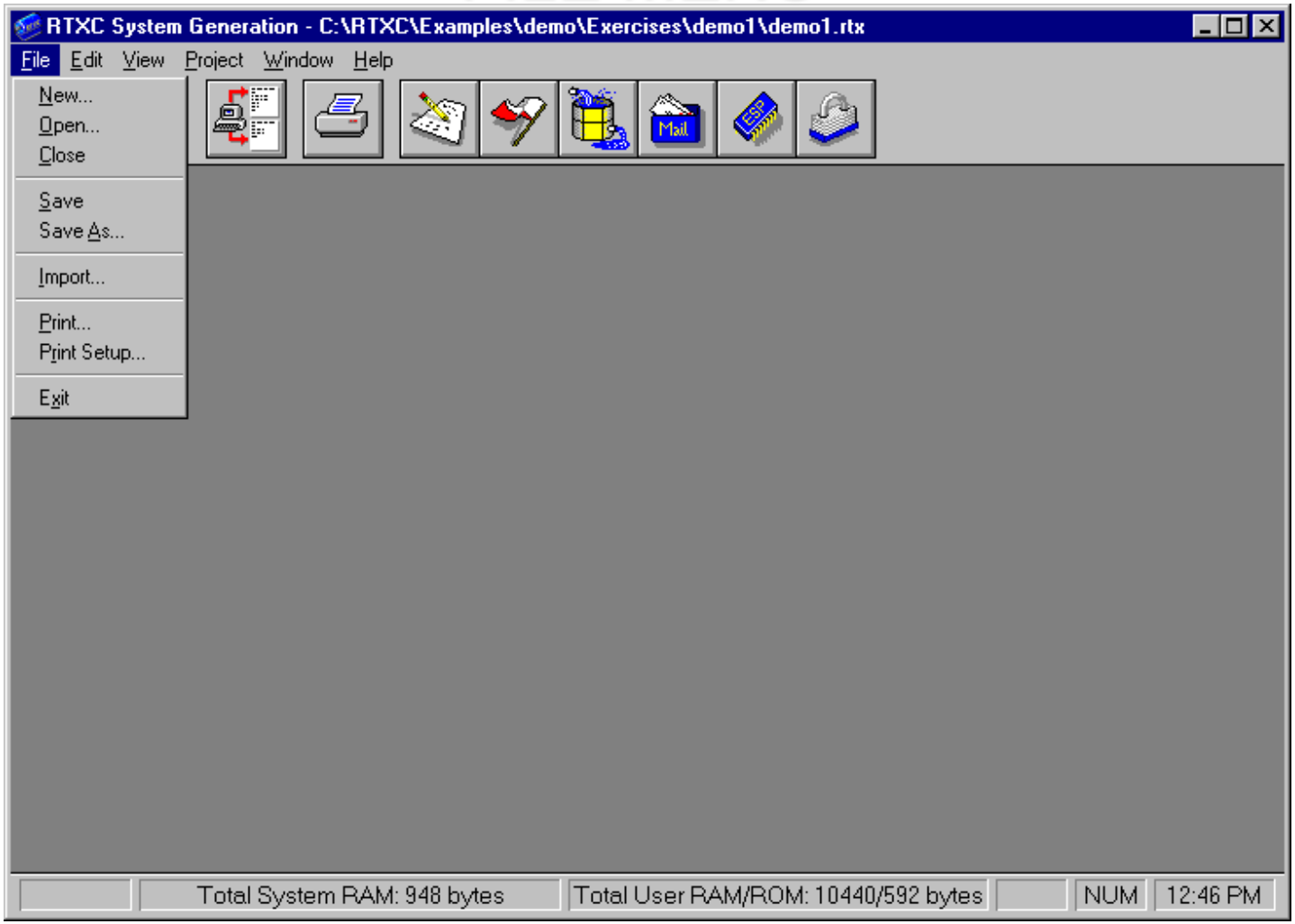


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

FILE MENU

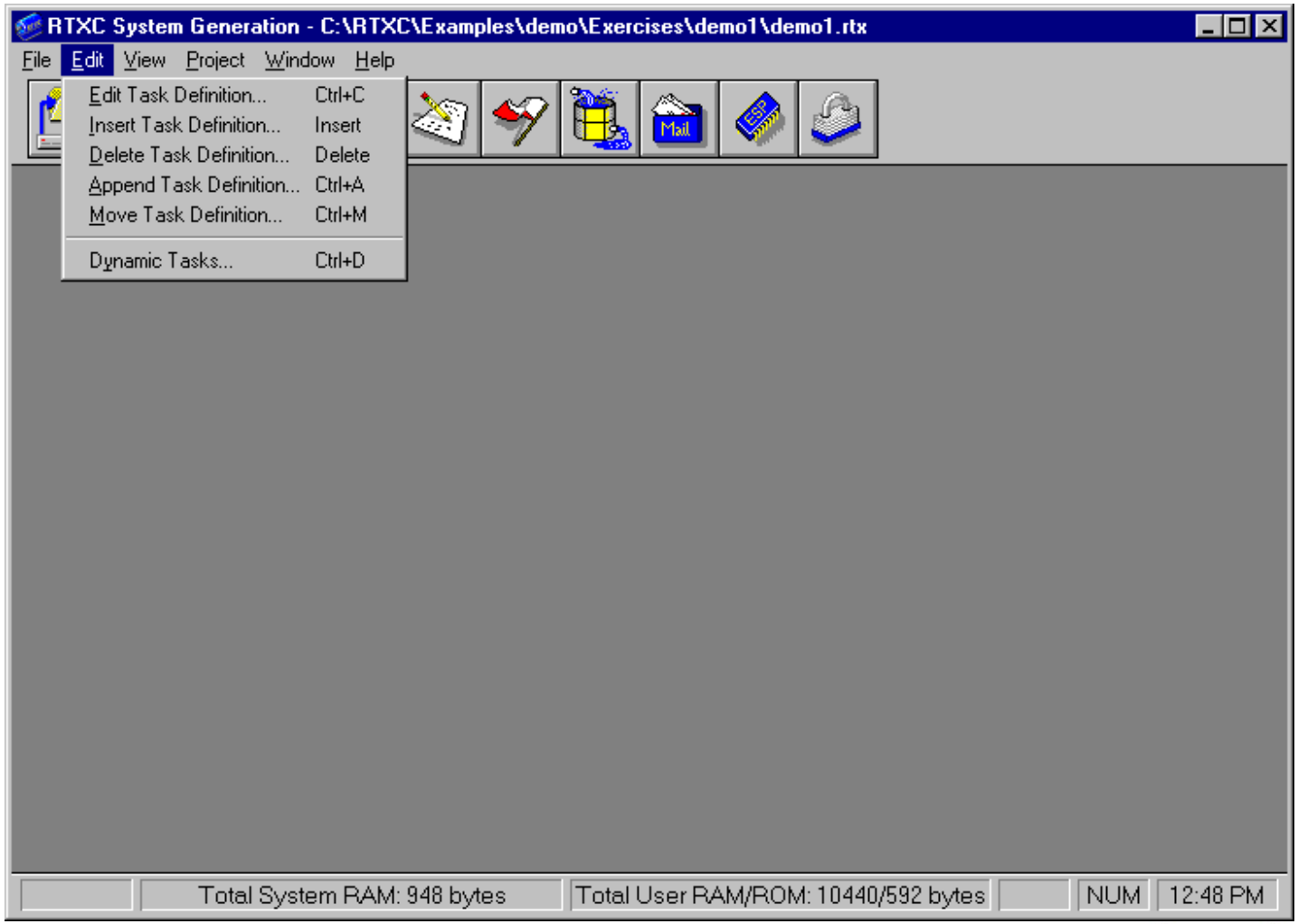


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

EDIT MENU

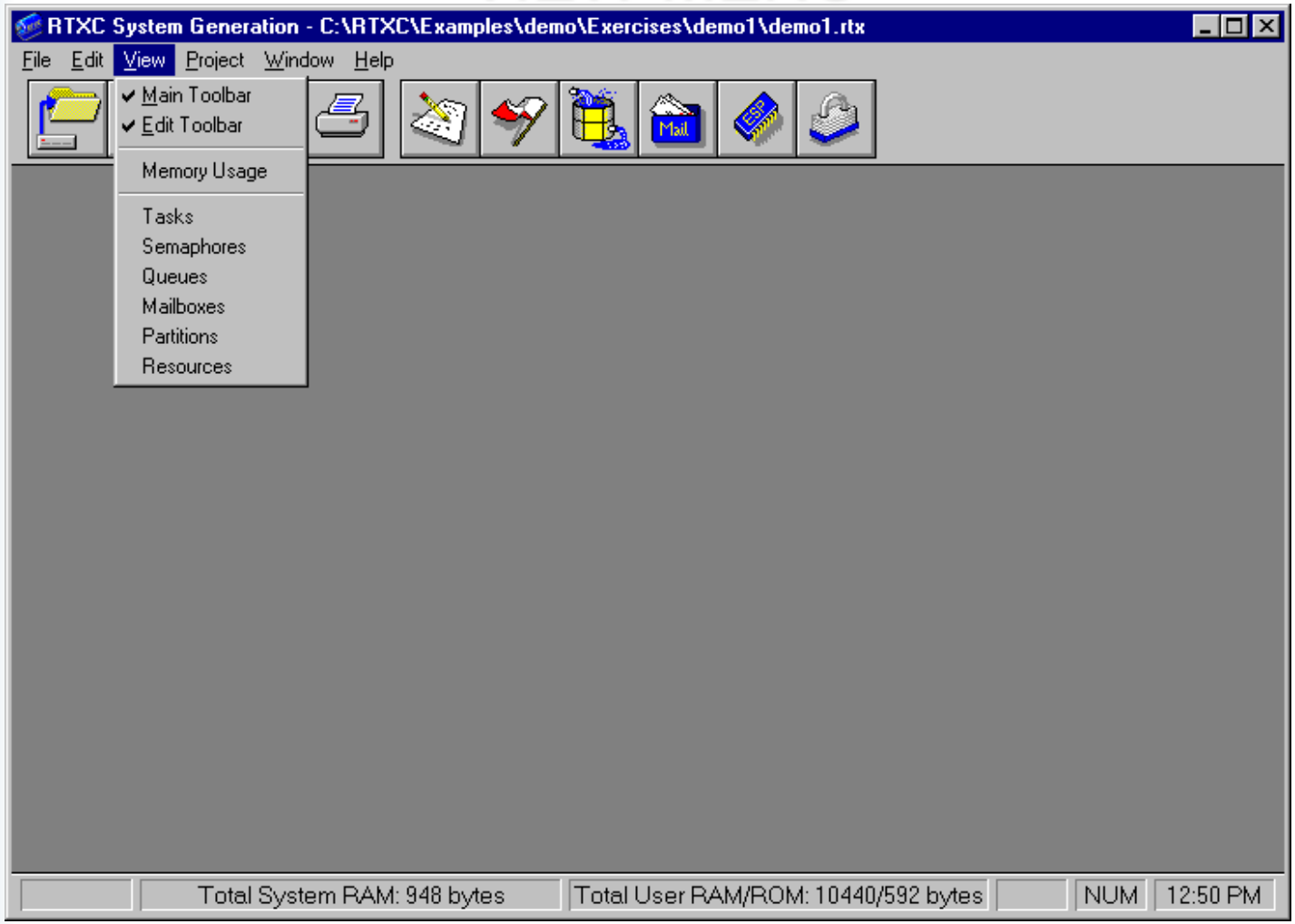


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen

VIEW MENU

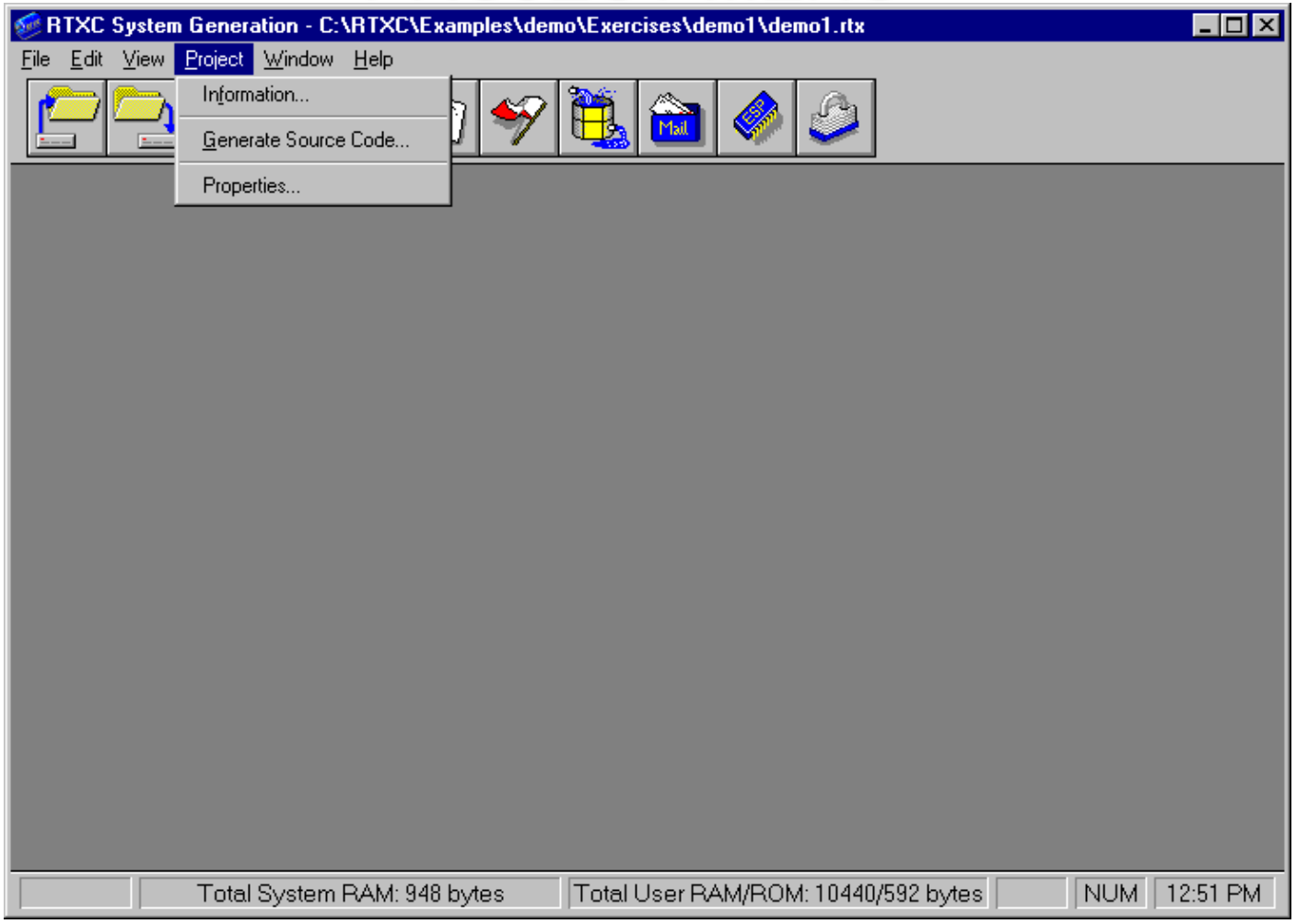


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

PROJECT MENU

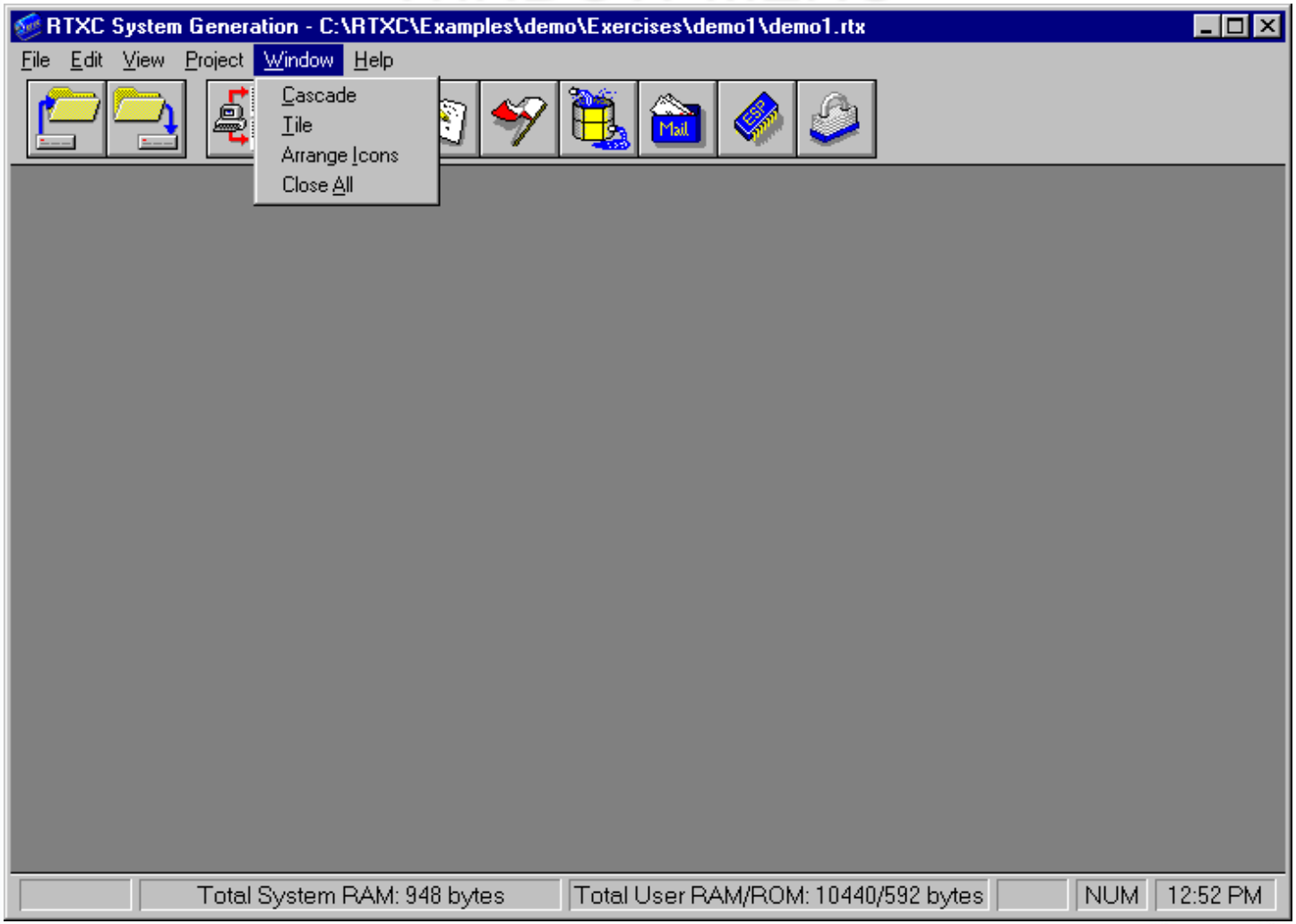


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen

WINDOW MENU

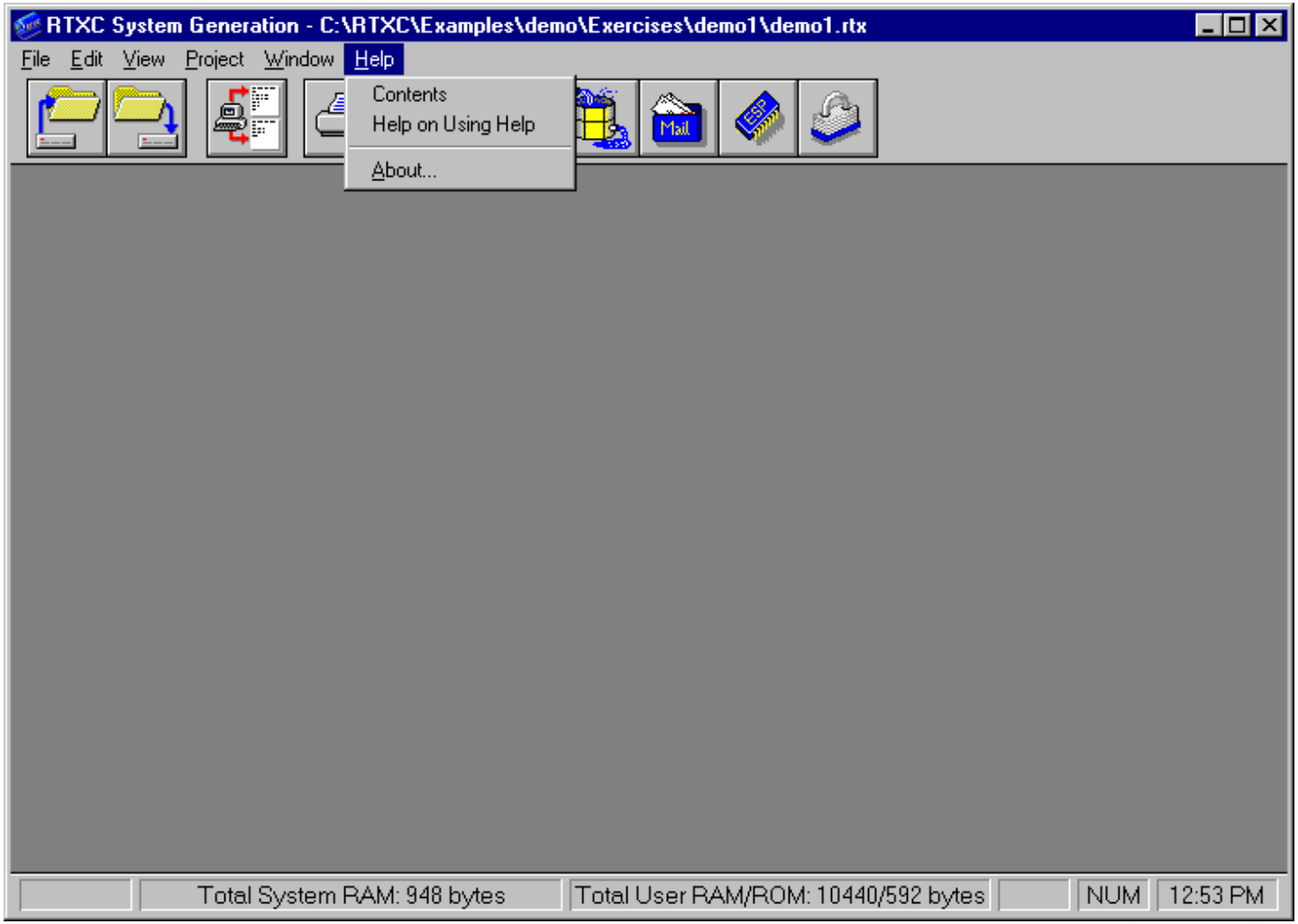


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen

HELP MENU

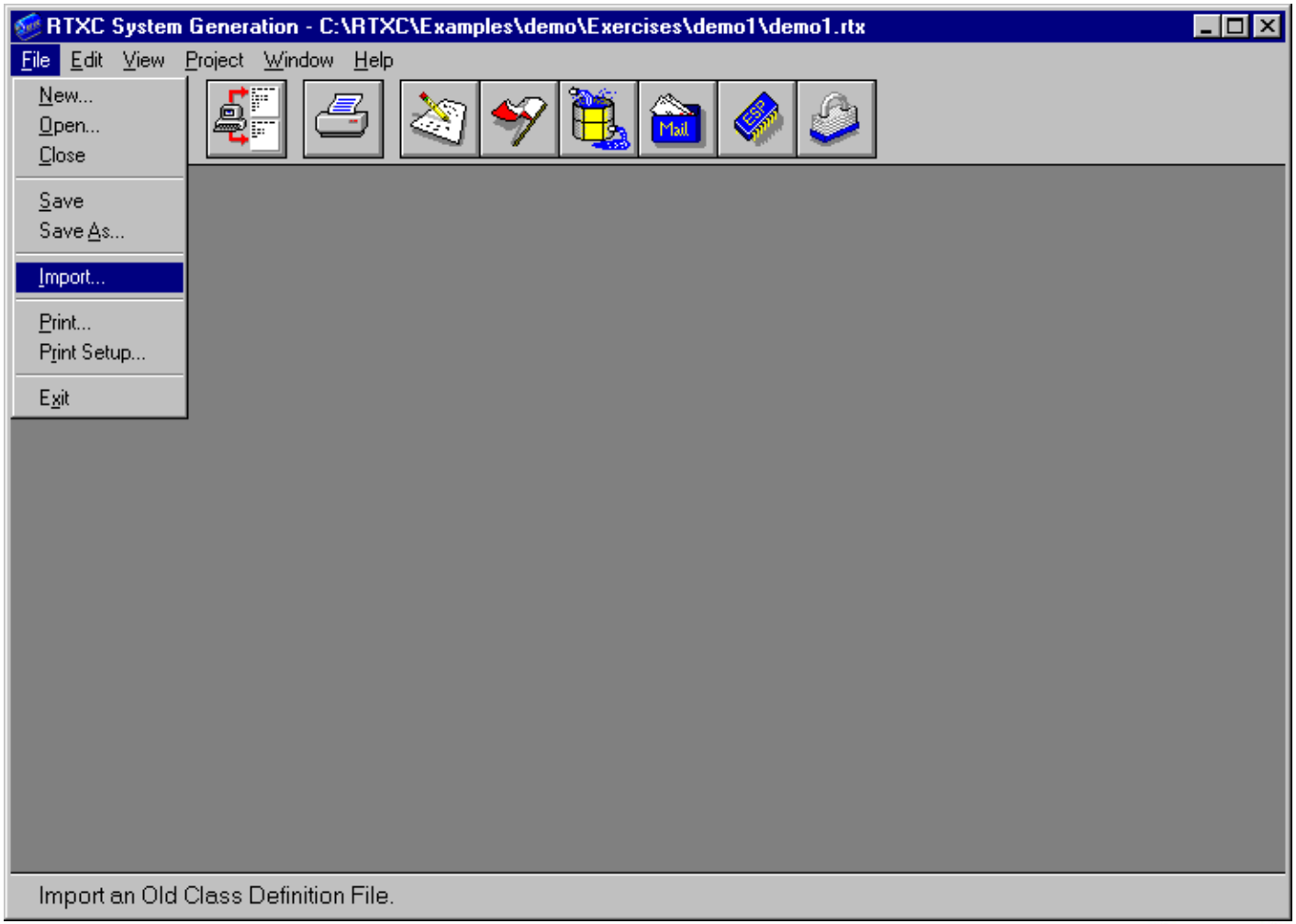


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

LOADING A TASK DEFINITION FILE

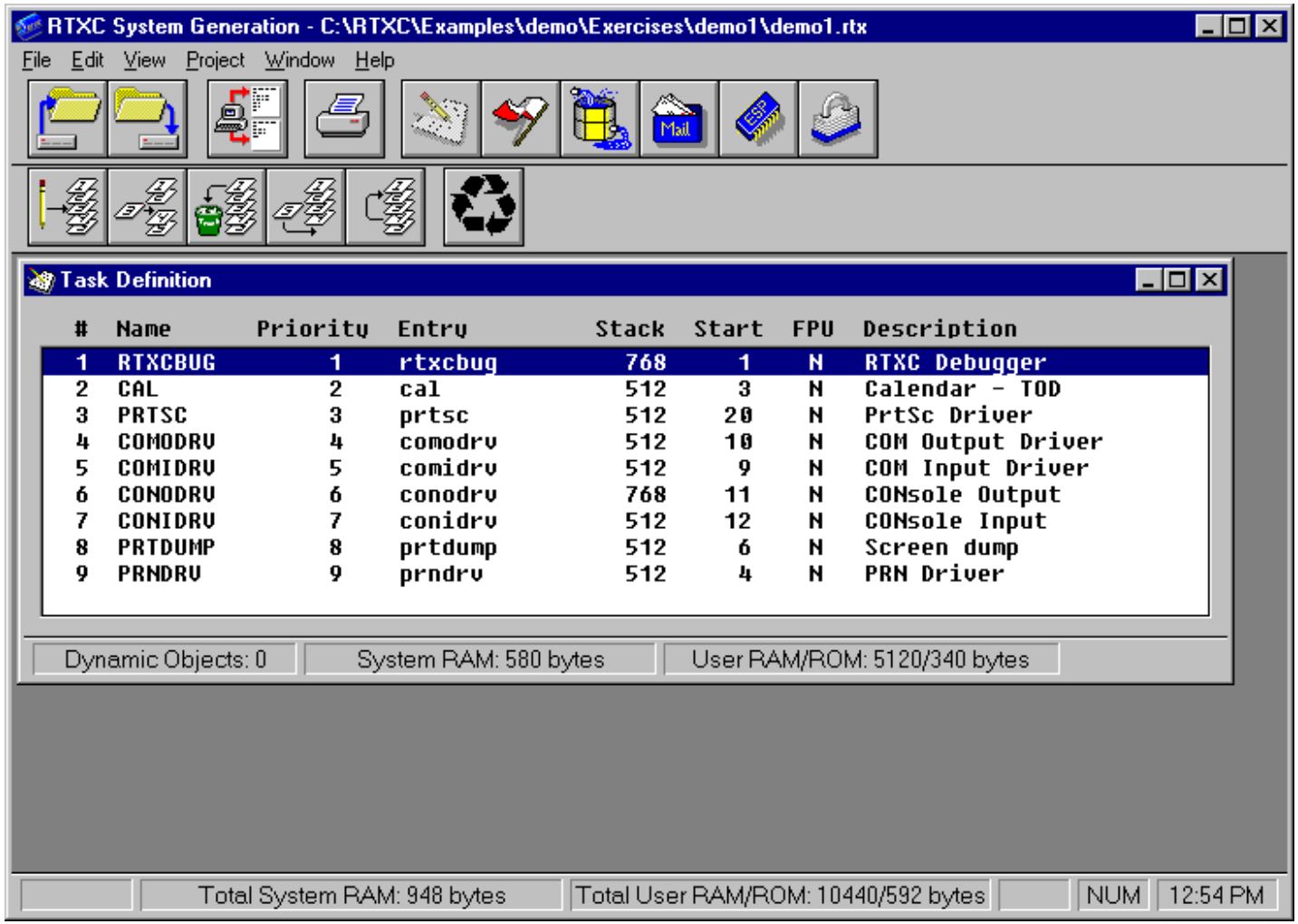


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen

TASK DEFINITION WINDOW

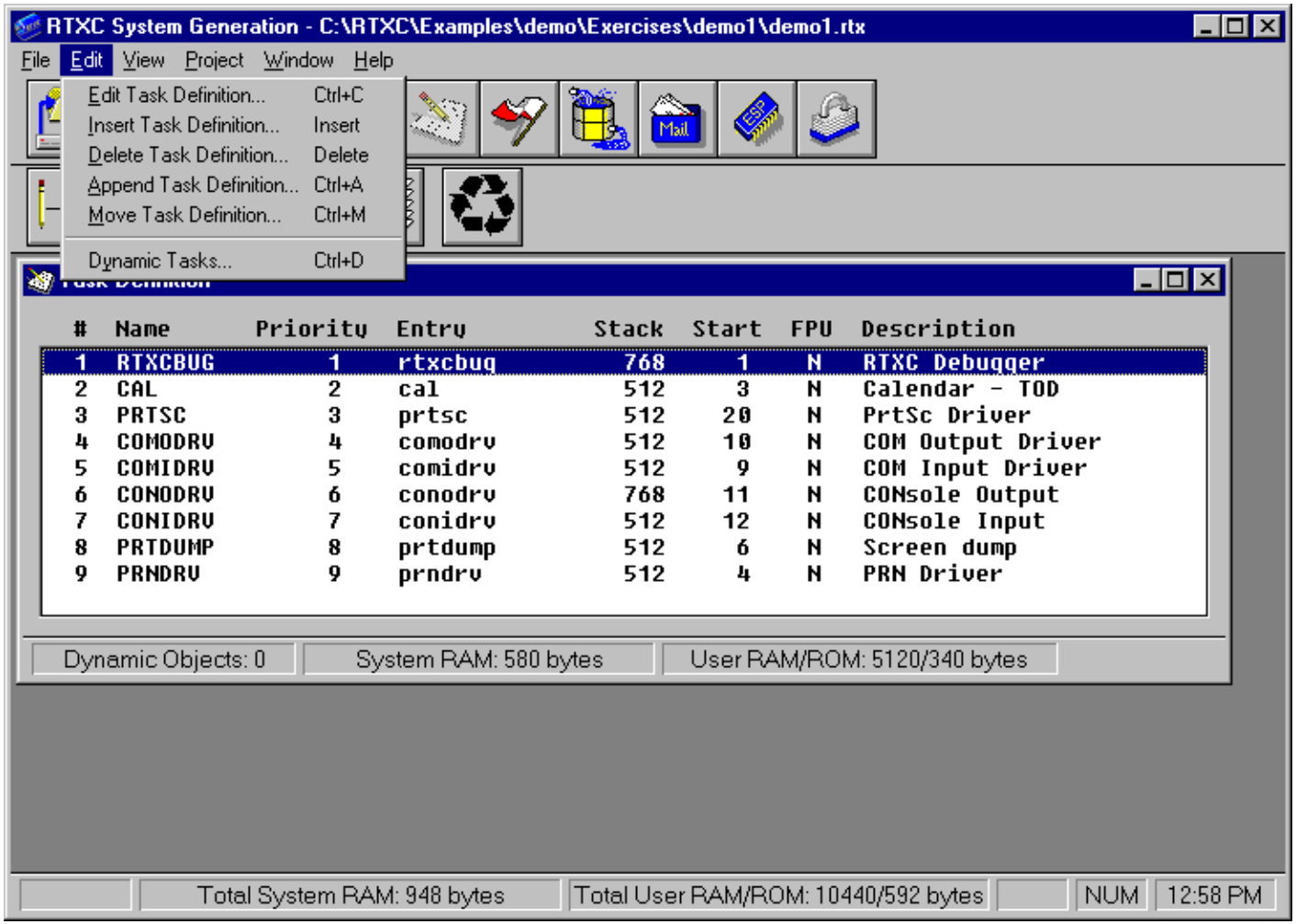


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

TASK EDIT MENU

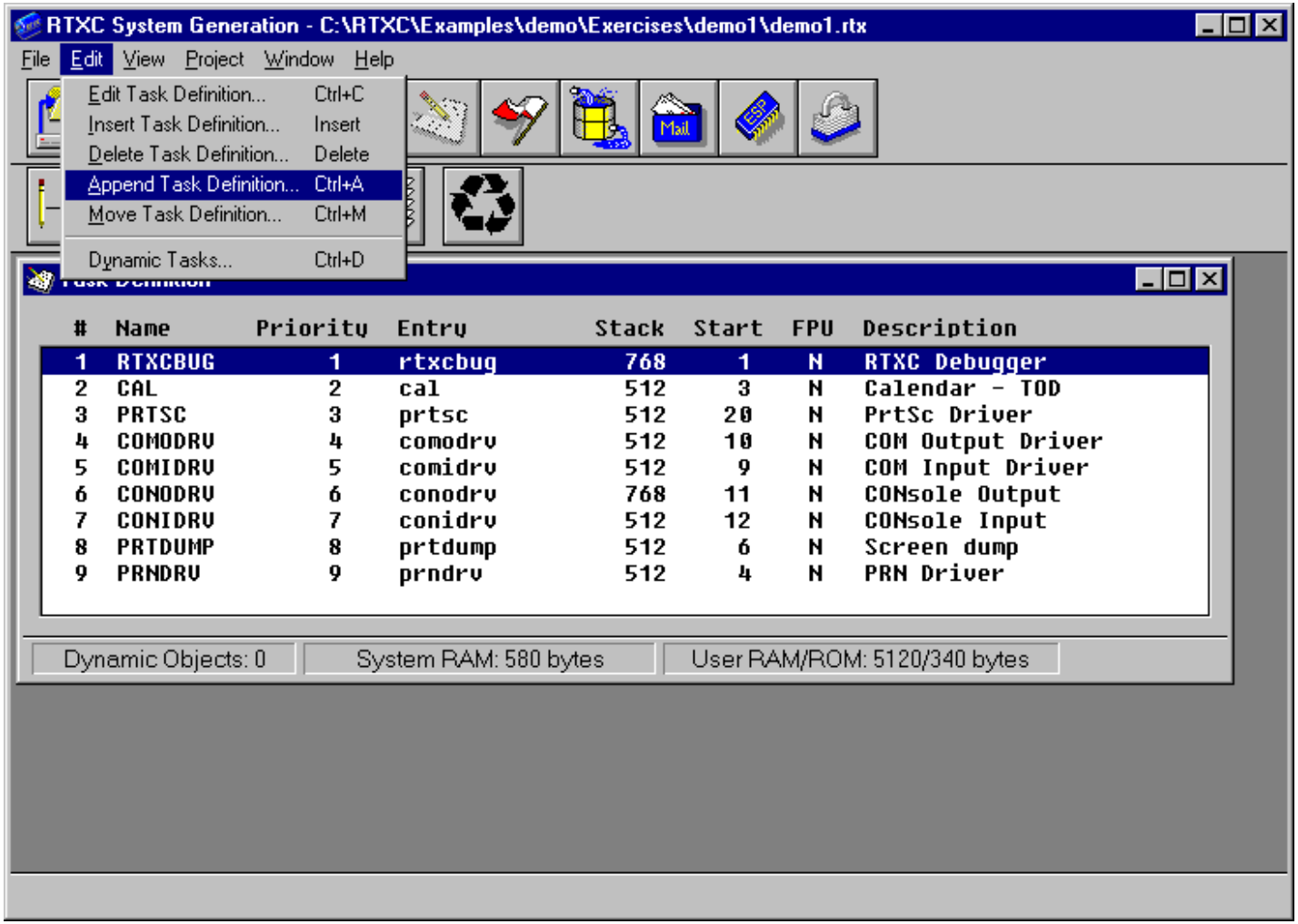


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

APPENDING A NEW TASK

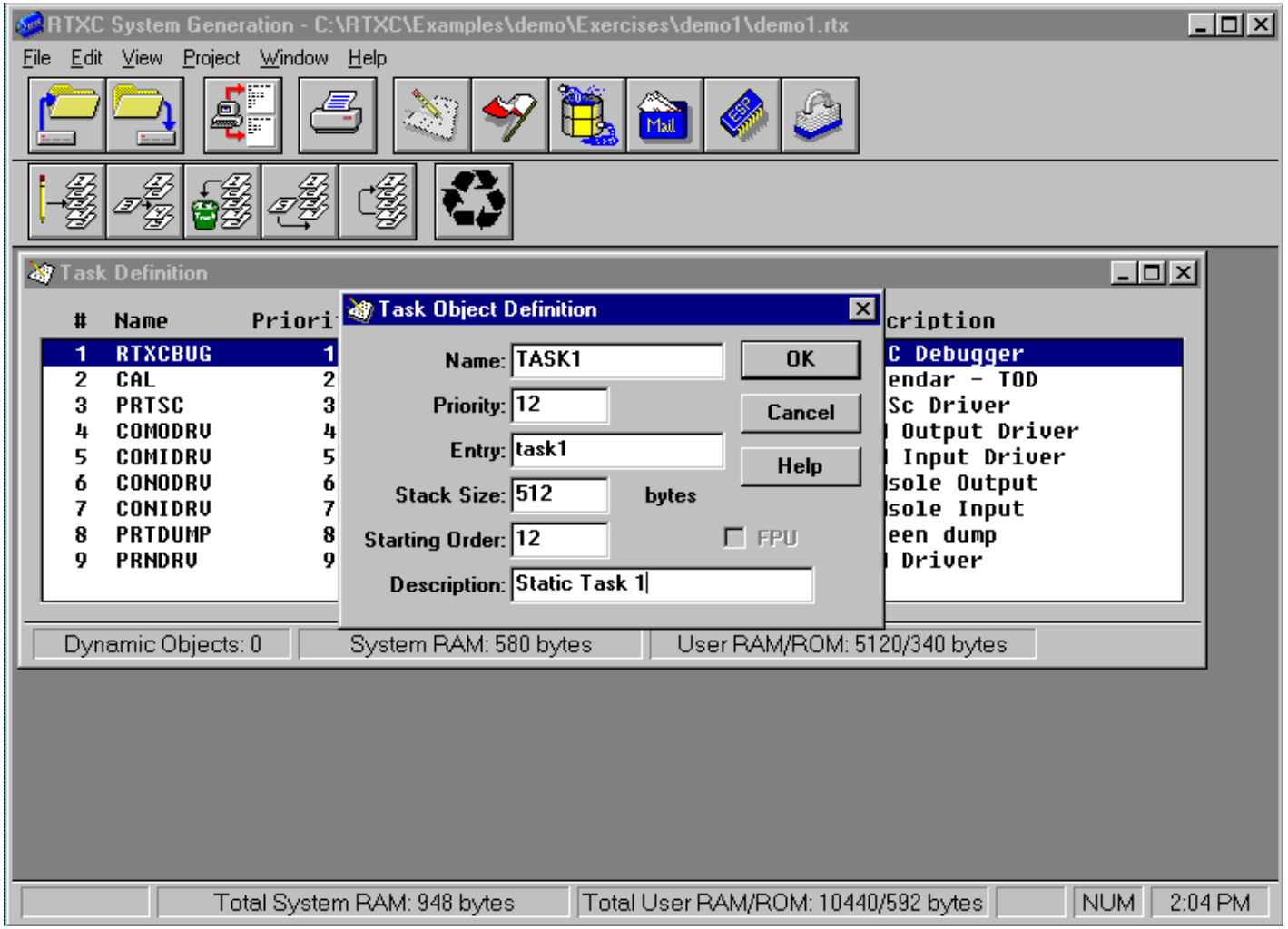


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

APPENDING A NEW TASK

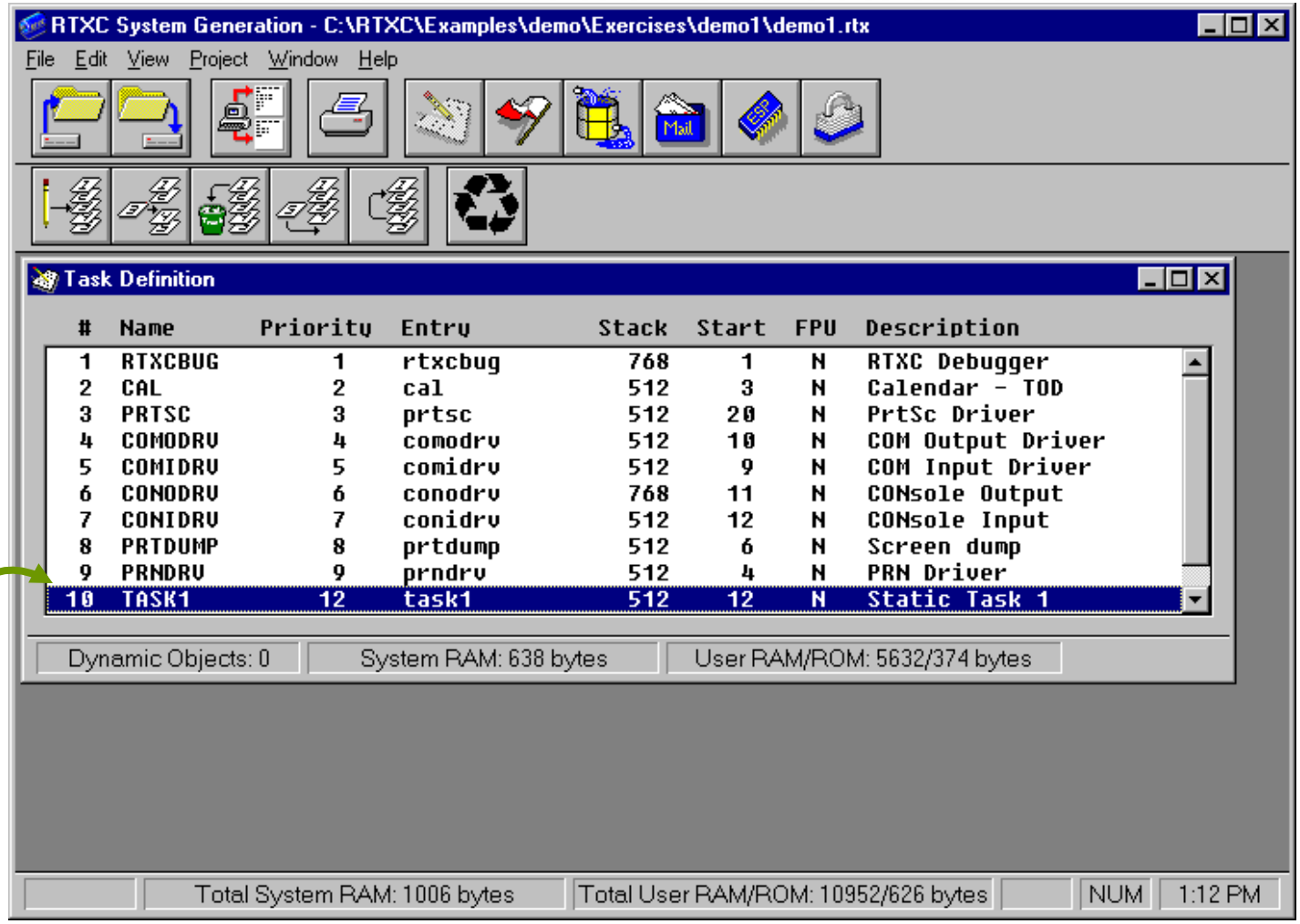


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

TASK DEFINITION WINDOW



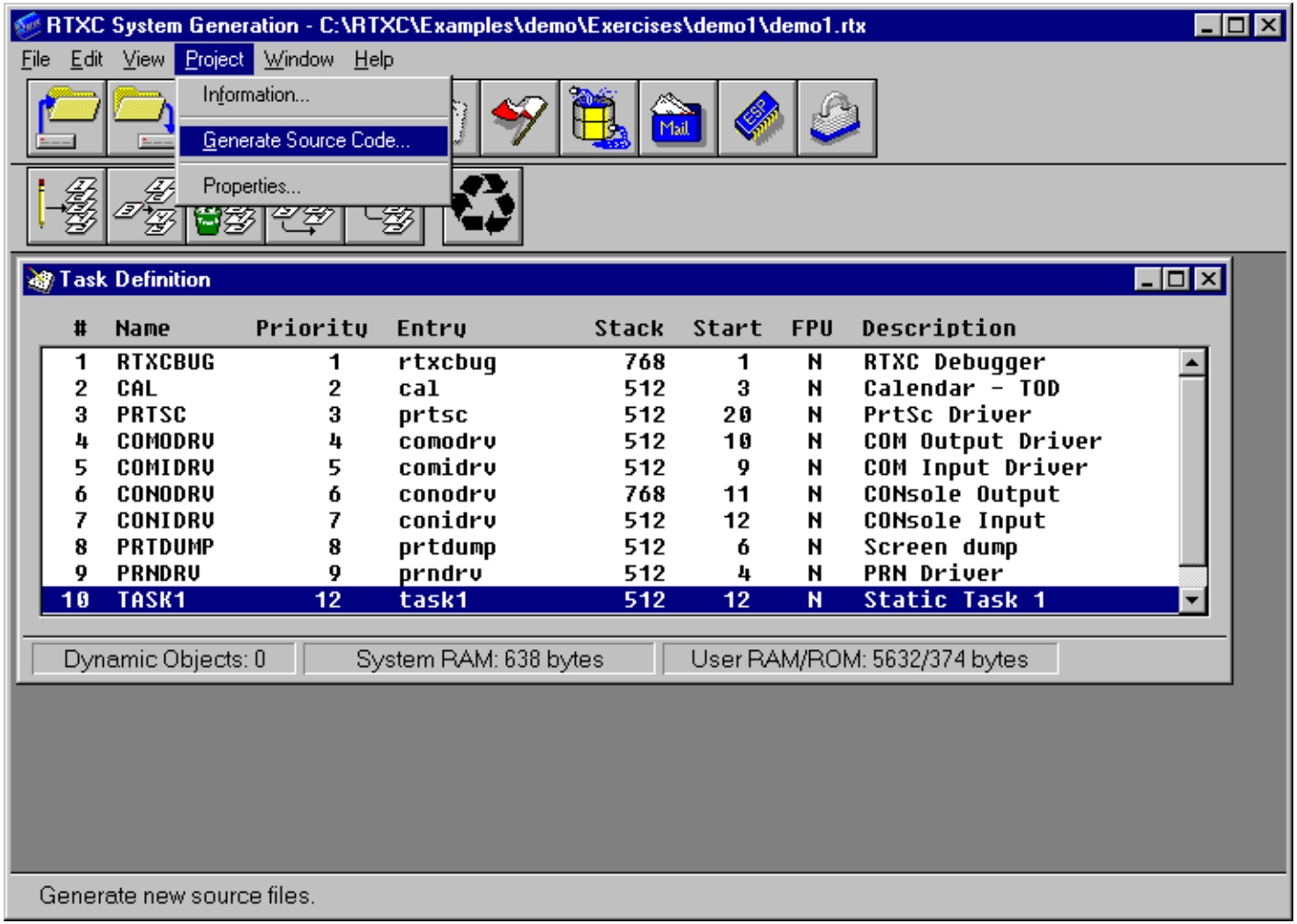
Appended Task

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

GENERATING THE DEFINITION SOURCE CODE

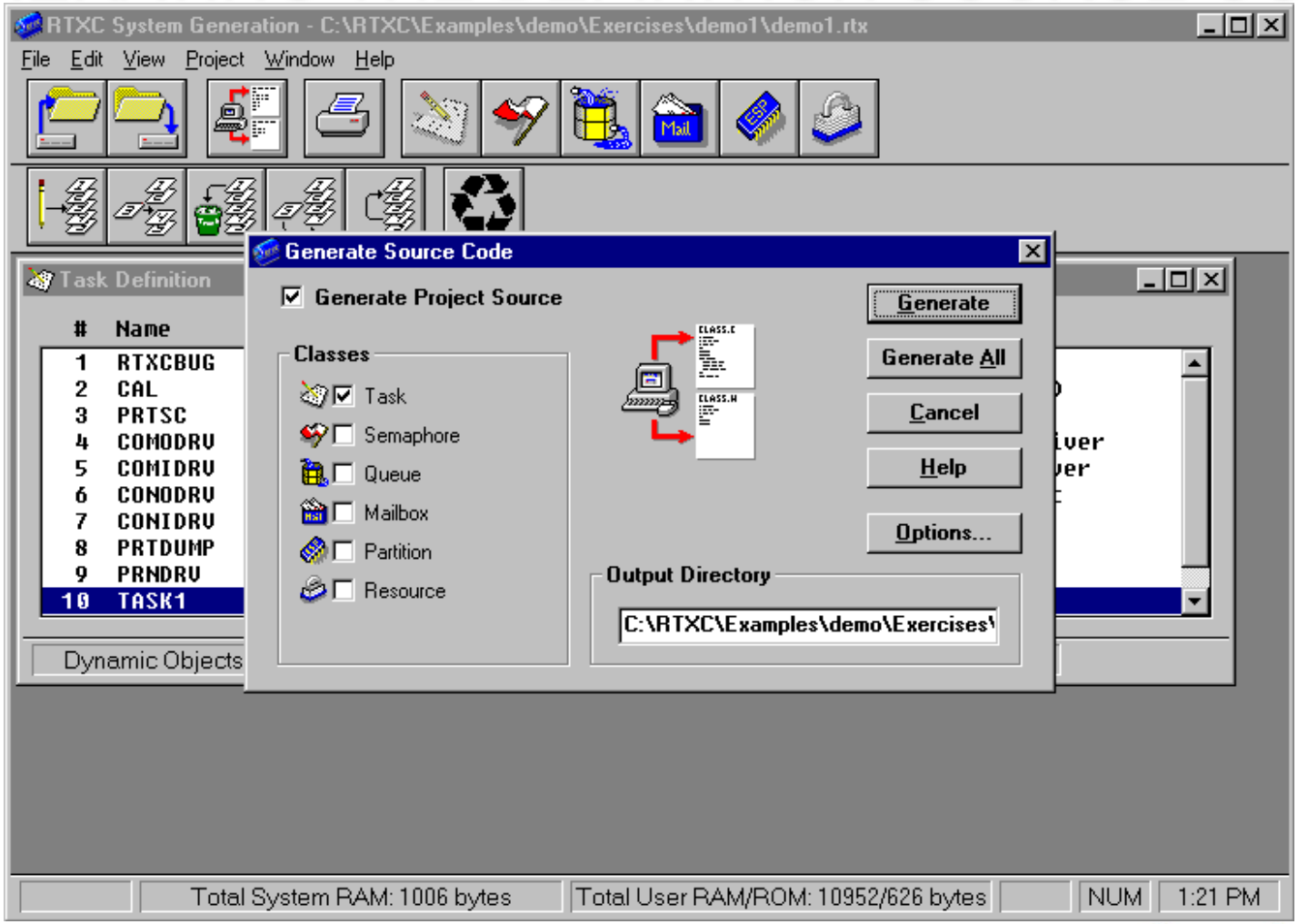


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

GENERATING THE DEFINITION SOURCE CODE



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SYSgen

MODULE NAMES (xxxx)

- TASK — Task control block definitions
- SEMA — Semaphore definitions
- RES — Resource definitions
- MBOX — Mailbox definitions
- QUEUE — Queue definitions
- PART — Memory Partition definitions
- CLOCK — Clock definitions

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen .h files

CTASK.H

```
#define NTASKS  10

#define DNTASKS 6

#define RTXCBUG   1 /* CBUG debugger */
#define CONAIDRV  2 /* Console A Input Task */
#define CONAODRV  3 /* Console A Output Tas */
#define CONBIDRV  4 /* Console B Input Task */
#define CONBODRV  5 /* Console B Output Tas */
#define EXAMPLE1  6 /* Example Task 1 */
#define EXAMPLE2  7 /* Example Task 2 */
#define FPUTASK1  8 /* Floating Point Task */
#define FPUTASK2  9 /* Floating Point Task */
#define TASK1     10 /* Task1 */
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SYSgen SUMMARY

- For best results follow this procedure:
  - Open- a project file
  - Edit- make the changes to the kernel objects
  - Save- save the project file information
  - Generate- generate the source code. (.c and .h files)

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

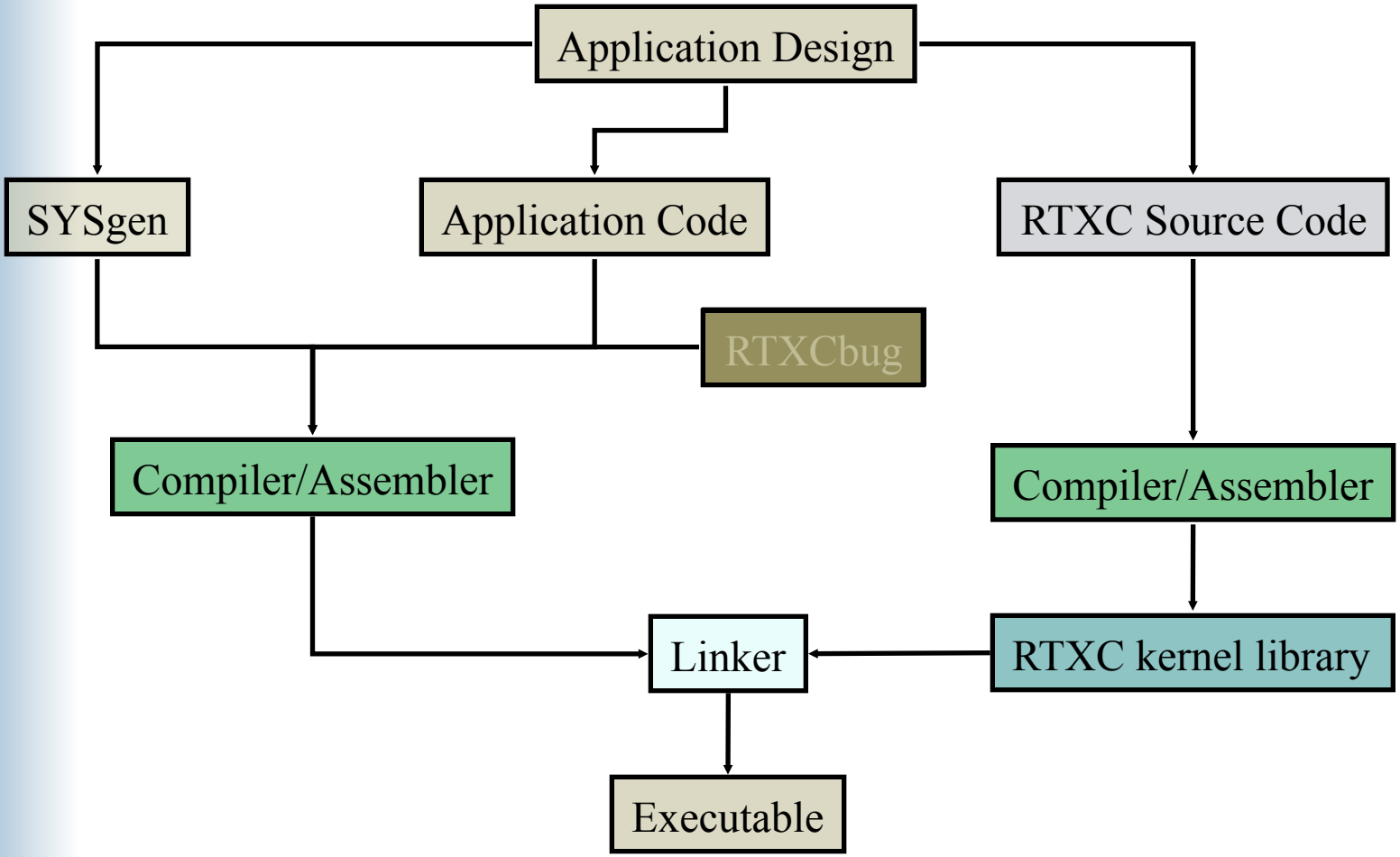
RTXCbug

SYSTEM LEVEL DEBUG FACILITY

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

APPLICATION DEVELOPMENT PROCESS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug

SYSTEM LEVEL DEBUG FACILITY

- Symbolic system level inquiry, display and control
- Invoked manually or via a program
- Provides coherent snapshots of internal system elements
- Limited task control capabilities
- Symbolic element names used for displays and dialog

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXCbug

- User defined debug console device via RTXCgen
  - System console
  - Serial port
- Operates as a task (usually highest priority)
  - Uses no system resources until invoked
  - Freezes active timers
  - Blocks all tasks except the debug console driver
  - Allows interrupt servicing

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug

MAIN MENU

```
** RTXCbug - RTXC v3.2a xxxx vx.x Date

  K - RTXC
  G - Go to Multitasking Mode
  X - Exit RTXCbug

RTXCbug>
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug

HELP MENU

```
T - Tasks
M - Mailboxes
P - Partitions
Q - Queues
R - Resources
S - Semaphores
C - Clock/Timers
K - Stack Limits
Z - Zero Partition/Queue/Resource Statistics
$ - Enter Task Manager Mode
# - Task Registers
G - Go to Multitasking Mode
U - Return to Main Menu
X - Exit RTXCbug
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXCbug

SUMMARY

- Collects “high water mark” statistics
- Uses no resources until invoked
- Can be used to launch tasks individually to debug them
- The information collected may be used to redesign and re-code the application

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

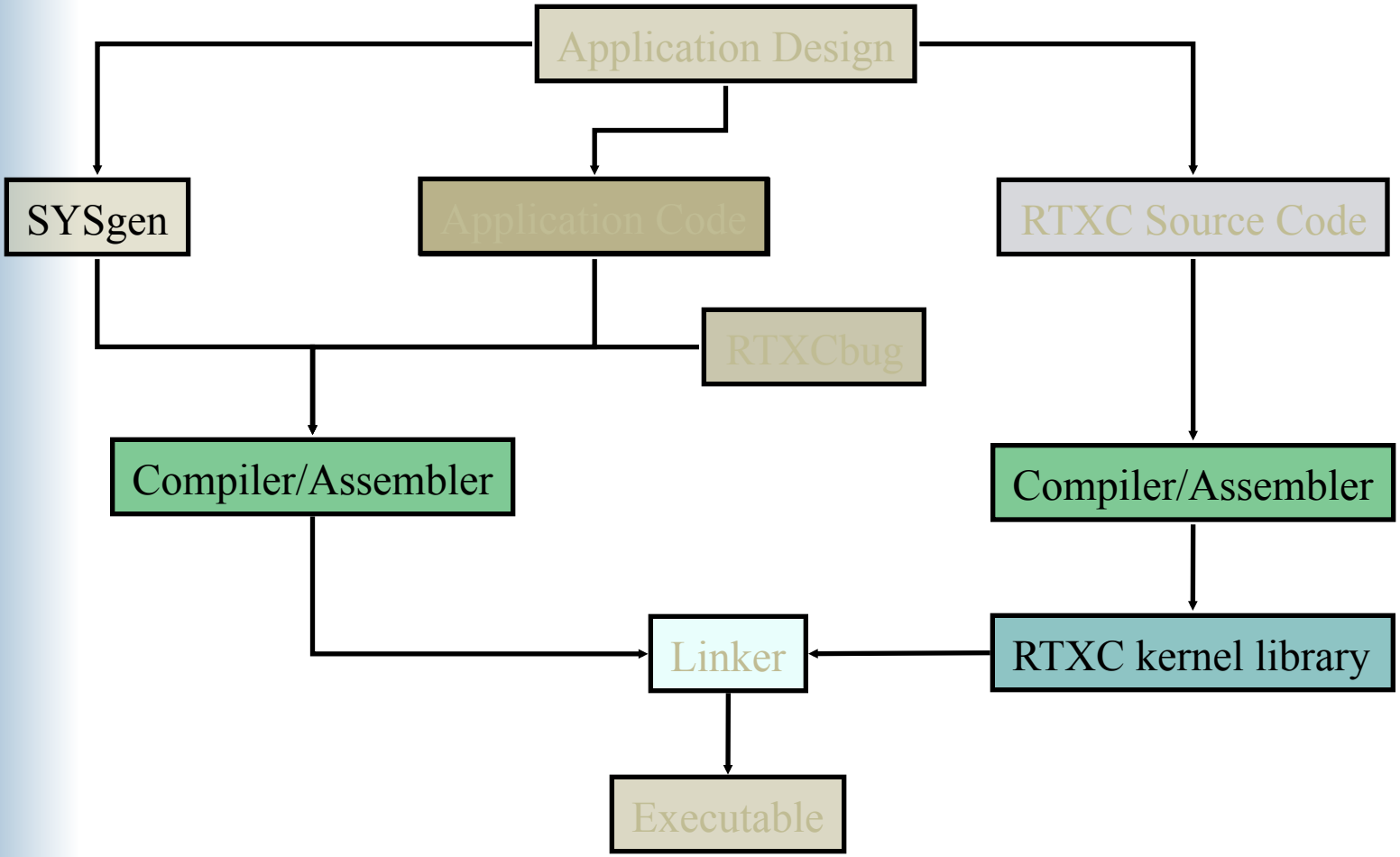
Quadros™ Systems Inc.

APPLICATION CODE

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

APPLICATION DEVELOPMENT PROCESS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

KERNEL SERVICES

GENERAL CONVENTIONS

- Names begin with `KS_`
- Name format is `KS_verb[noun][suffix](object, ...)`
- Suffixes:
  - example: `KS_sendw`, `KS_sendt`, `KS_waitm`
    - w - unconditionally waits
    - t - waits with timeout values
    - m - waits on multiple
- A task argument of 0 means Current Task
- def services define; inq services inquire
  - example: `KS_defpriority`, `KS_inqpriority`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

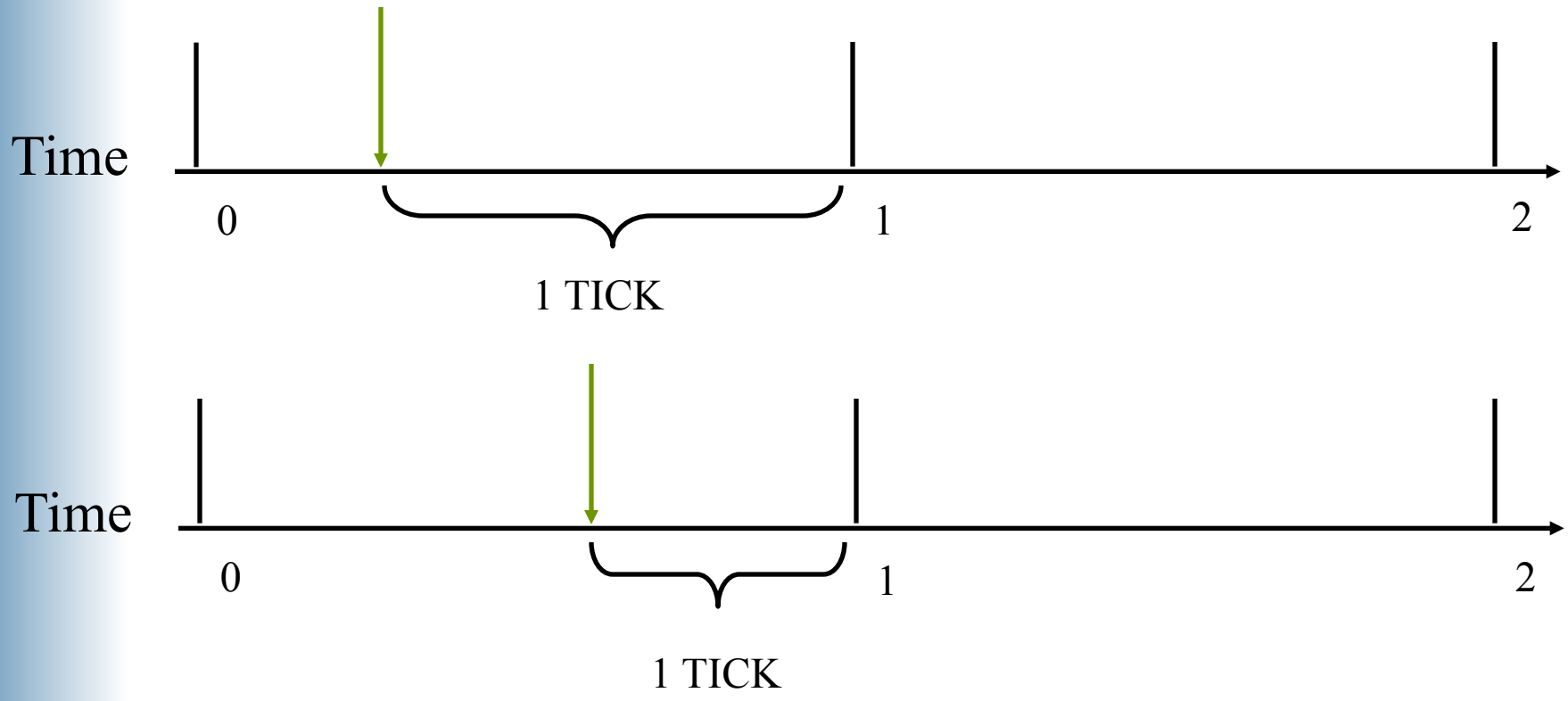
RTXC TIME

- All time is measured in units called “ticks”
- The duration of 1 tick depends on the frequency of the system interval timer
- Units of ticks are milliseconds
- The smallest tick duration supported by RTXCgen is 1 ms
- The file cclock.h defines the duration of 1 tick with the constant CLKTICK
- To convert from real time to system time express real time in ms and divide by CLKTICK
  - 5 sec = 5000 ms = 5000/CLKTICK

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

INACCURACIES WITH SINGLE TICK DURATIONS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

APPLICATION CODE SUMMARY

- Consists of C source files
- Can have multiple modules
- Will interact with files generated by SYSgen
  - Must include those SYSgen (.h) files needed
- Use Kernel Service calls to manipulate Kernel Objects
- Links with RTXC library

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TRAINING COURSE

PRACTICAL EXERCISES

- **Exercise #1**
  - Familiarization with SYSgen
  - Defining a task and a resource
  - Getting an application to run
  - Observe dynamic task priorities

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TRAINING

UNIT 3

Tasks and Memory Partitions

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TASKS

FUNDAMENTAL PROGRAM MODULE

- Written as a C function - void return and void arguments
- Can be instantiated dynamically or statically
- Usually performs a limited function (Spark control, Fuel injection etc.)
- Each task requires its own stack
- Task priority defined at System Generation
- Task priority variable during runtime
- Task status and context automatically maintained by RTXC Scheduler

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC TASKS

TWO FORMS: RUN AND TERMINATE

```
void taskname(void)
{
      ...Data declarations
      ...Task initialization
      ...Task operations

      KS_terminate(SELF);
}
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC TASKS

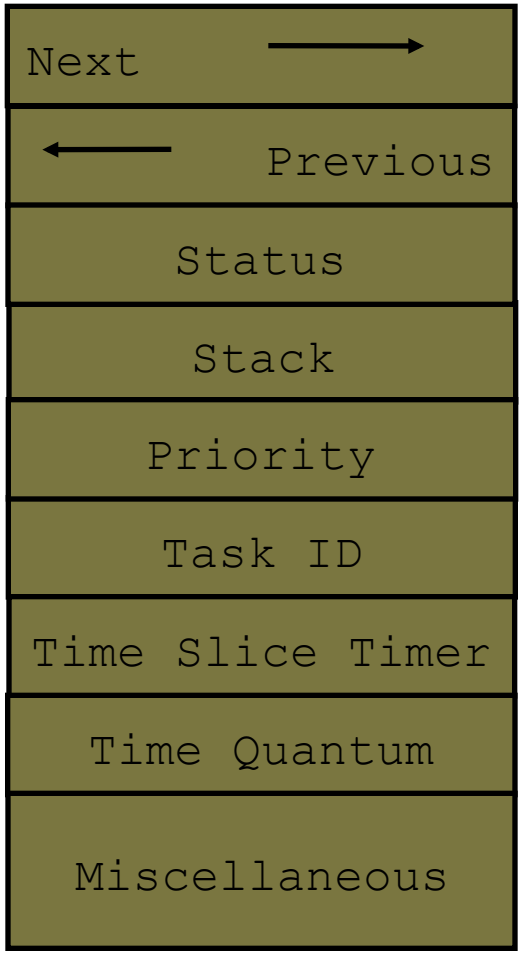
TWO FORMS: RUN FOREVER

```
void taskname(void)
{
        ... Data declarations
        ...Task initialization

        for (;;)
        {
        ...Task operations
        }
}
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

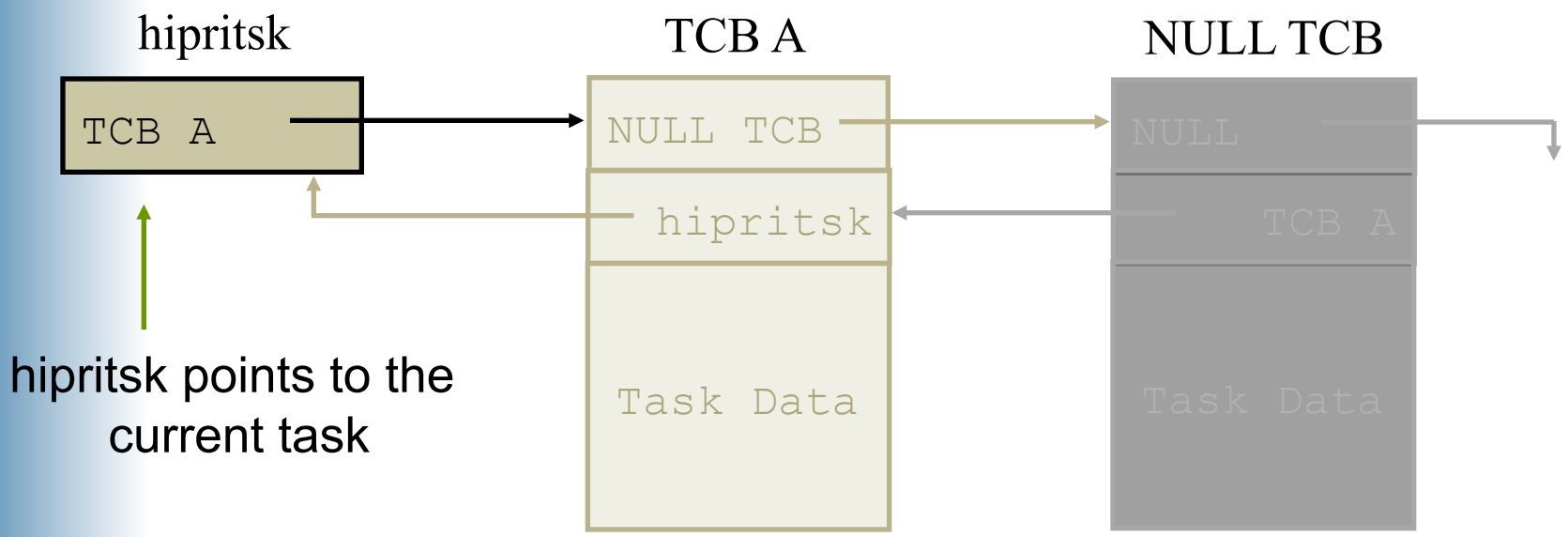
TASK CONTROL BLOCK



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

READY LIST LINKAGE

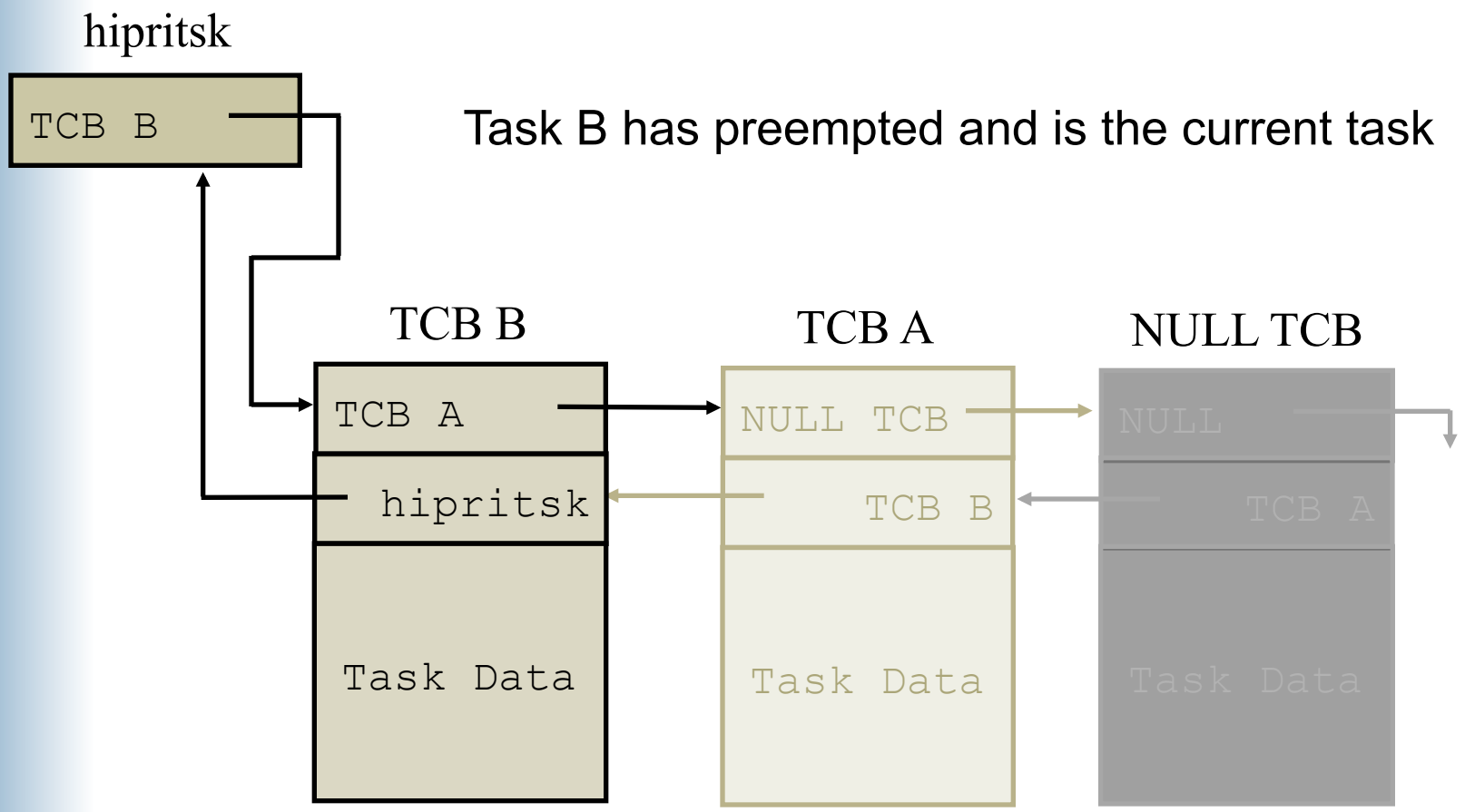


Task A is the current task
Note the use of the NULL task

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

READY LIST LINKAGE

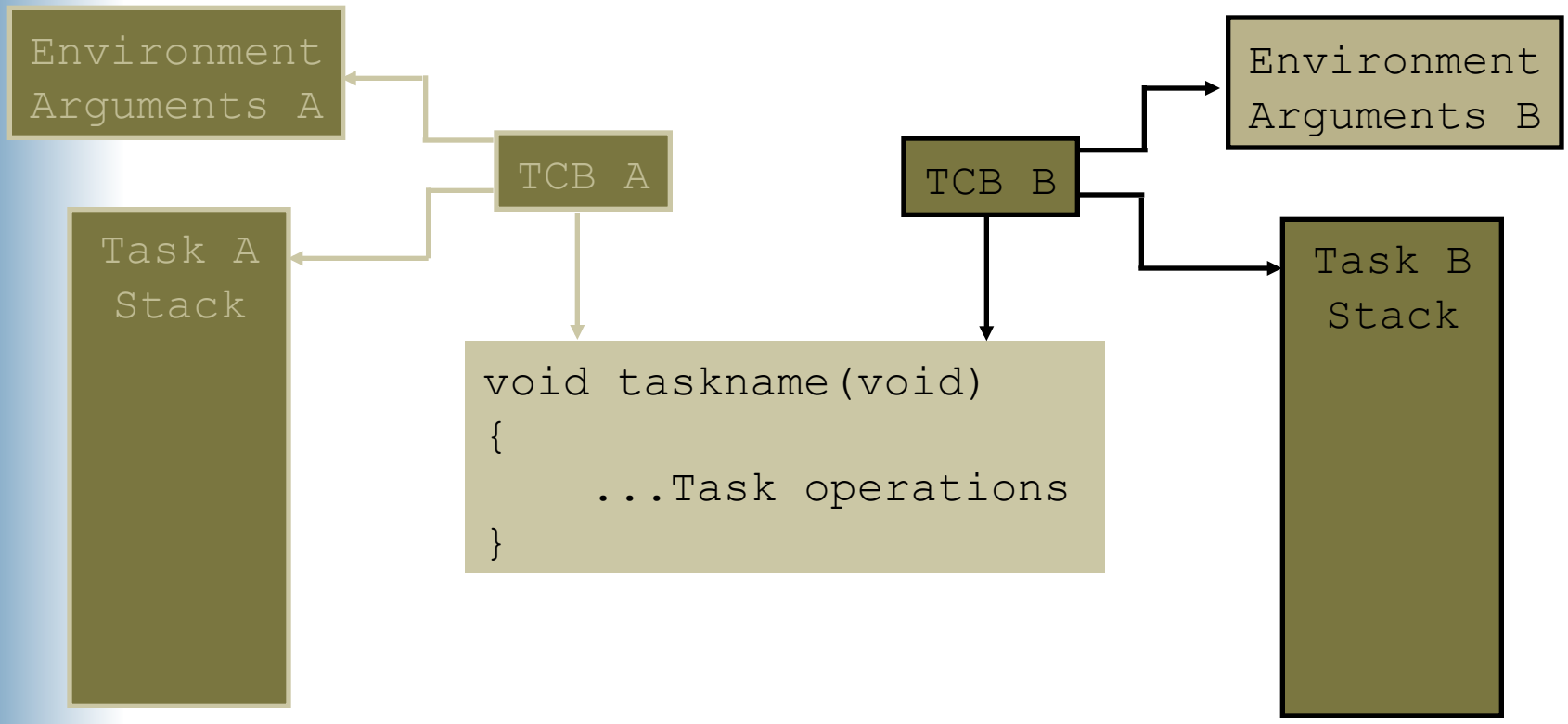


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC TASKS

SHARING TASK CODE



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

TASK KERNEL SERVICES

- `KS_execute(task)`
  - Make `task` READY for execution beginning at its starting address
- `KS_terminate(task)`
  - Terminate operation of `task`
- `KS_resume(task)`
  - Resume suspended `task`, unless otherwise blocked
- `KS_suspend(task)`
  - Suspend operation of `task`
- `KS_delay(task,ticks)`
  - Block `task` for a period of time specified by `ticks` then unblock `task` when delay expires

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

TASK KERNEL SERVICES

- `KS_defpriority(task, priority)`
  - Change the priority of `task` to `priority`
- `KS_inqpriority(task)`
  - Get `task`'s priority
- `KS_defslice(task, ticks)`
  - Define a `task`'s time-slice time quantum
- `KS_inqslice(task)`
  - Get `task`'s time-slice time quantum
- `KS_yield(void)`
  - Yield control of the CPU to another task, used with round-robin scheduling

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

TASK KERNEL SERVICES

- `KS_alloc_task(void)`
  - Allocate a TCB from the pool of dynamic TCBs
- `KS_deftask(task, priority, stack *, stack size, entry)`
  - Define attributes of `task`
- `KS_deftask_arg(task, envarg *)`
  - Define the environment arguments of `task`
- `KS_inqtask_arg(task)`
  - Get address of `task`'s environment arguments

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

TASK KERNEL SERVICES

- `KS_inqtask(void)`
  - Get `task` number of Current Task
- `KS_block(start, end)`
  - Blocks range of tasks from `start` to `end`
- `KS_unblock(start,end)`
  - Unblocks range of tasks from `start` to `end`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXCbug TASK INFORMATION

```
** Task Snapshot **
  #   Name  Priority  State
  1 RTXCBUG     1     READY
  2 PRTSC       2     -READY
  3 CONODRV     9     READY
  4 CONIDRV     8     QueueEmpty CONIQ
  5 HISTASK    12     INACTIVE
  6 COMODRV    10     QueueEmpty COMOQ
  7 DINP       11     Semaphore DINTSEMA SDINSEMA
  8 DEMO1      14     Semaphore DEMOSEM0 <QNE> DEMOQ
                      DEMOSEM2 <MBXNE> DEMOMBOX
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug TASK STATE DESCRIPTIONS

- The legitimate state descriptions are:

`INACTIVE`	Not executing
`READY`	Active and ready to run
`-READY`	Active but blocked by RTXCbug
`DELAY`	Blocked on a time delay
`SUSPENDED`	Suspended
`Semaphore`	Waiting on one or more events
`QueueEmpty`	Waiting on an empty queue
`QueueFull`	Waiting on a full queue
`Mailbox`	Waiting on an empty mailbox
`Resource`	Waiting on a busy resource
`Partition`	Waiting on an empty partition

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXCbug STACK SNAPSHOT

```
** Stack Snapshot **
  #    Task     Size  Used  Spare
  1 RTXCBUG     768   610   158
  2 CLKDRV      512   122   390
  3 PRTSC       512   124   388
  4 CONODRV     512   250   262
  5 DEMO1       256   150   106
  6 DINP        256   210    46

RTXC Kernel     256    68   188

Worst case interrupt nesting = 3
Worst case Signal List size =  2
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC MEMORY PARTITIONS

RAM MEMORY MANAGEMENT OBJECT

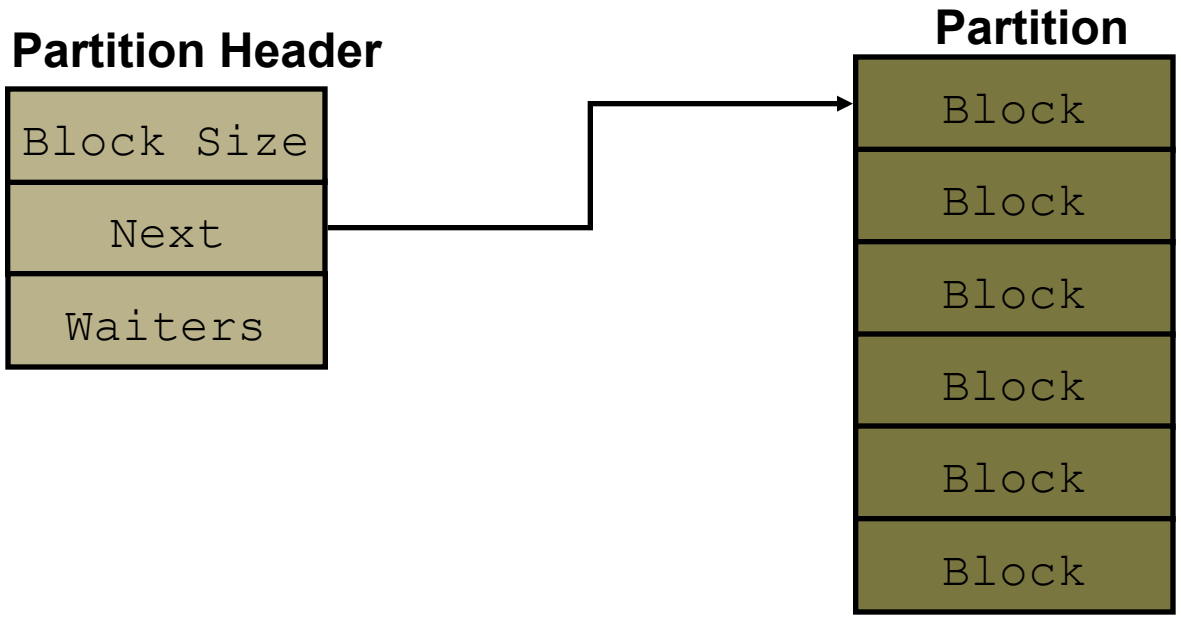
- Prevents fragmentation of memory
- Many Memory Partitions may be defined
- Also known as Memory Array Partitions - MAPs
- Memory Partitions may be defined statically and/or dynamically
- Memory Partitions defined globally
- Any task may allocate memory from any Memory Partition

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC MEMORY PARTITIONS

MEMORY PARTITION ORGANIZATION

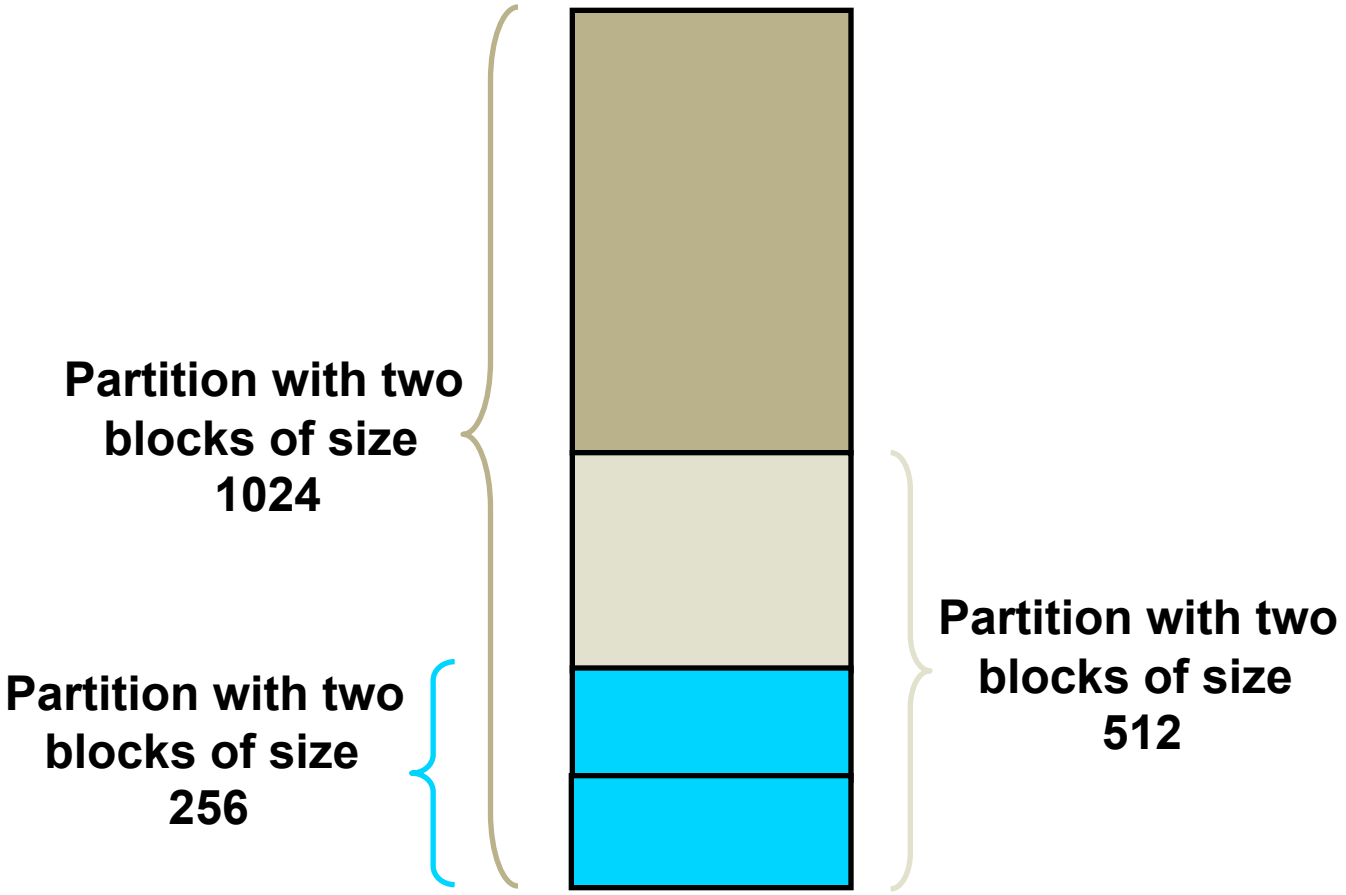


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC MEMORY PARTITIONS

SUBDIVISION OF PARTITIONS

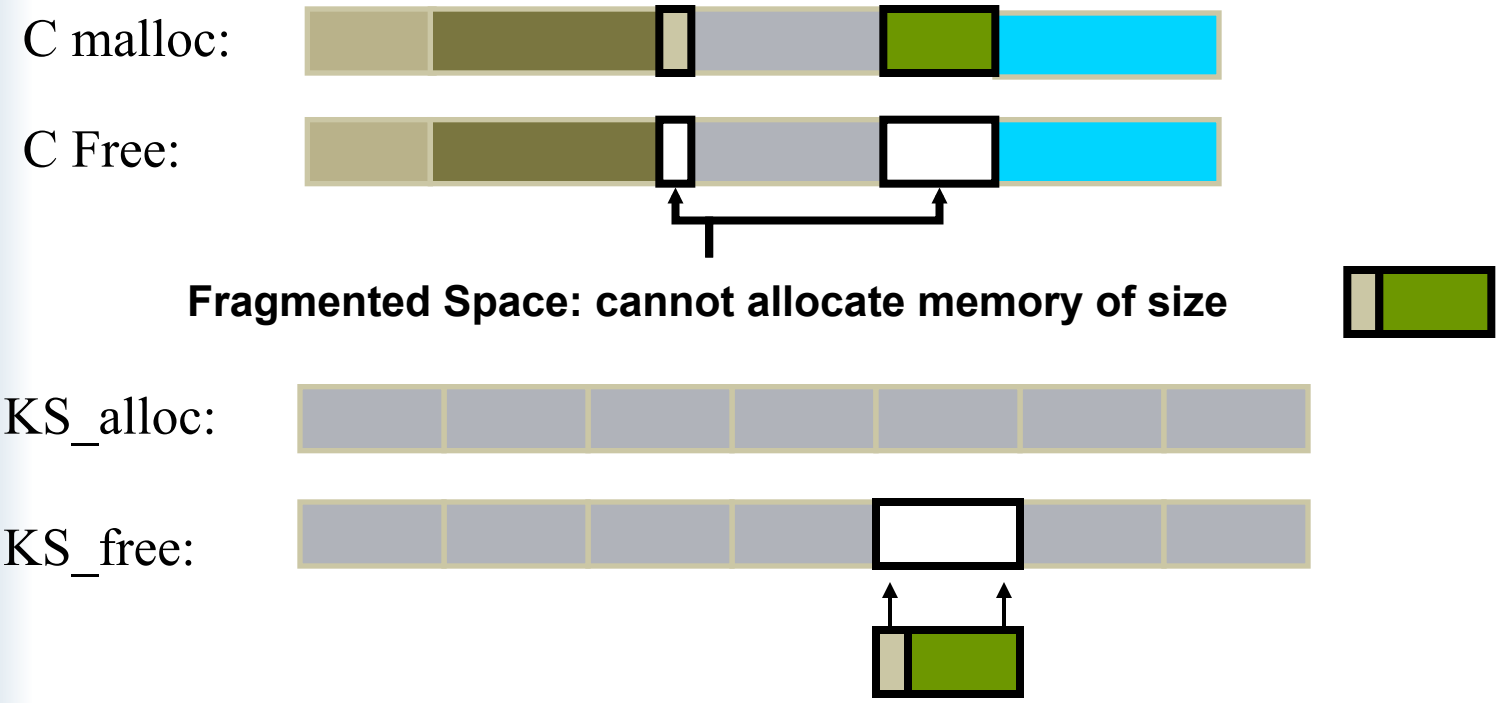


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC MEMORY PARTITIONS

AVOIDING FRAGMENTATION IN THE HEAP



Since blocks are the same size we have a perfect fit

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC MEMORY PARTITIONS

PARTITION KERNEL SERVICES

- `KS_alloc(map)`
  - Allocate a block of memory from `map` and return its address, returns `NULL` if partition is empty
- `KS_allocw(map)`
  - Same as alloc() but, if `map` is empty, wait until memory available
- `KS_alloct(map, ticks)`
  - Same as allocw() but limit duration of wait to a period of `ticks`
- `KS_free(map, address)`
  - Free the memory block at `address` to specified Memory Partition
- `KS_inqmap(map)`
  - Get the memory block size of `map`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC MEMORY PARTITIONS

PARTITION KERNEL SERVICES

- `KS_alloc_part(void)`
  - Allocate a dynamic Memory Partition control block and return its identifier, returns `NULL` if none available
- `KS_defpart(map, address, size, number)`
  - Define attributes of an existing or dynamically allocated partition, `map`, with a memory array located at `address` and containing `number` of blocks of size `size`
- `KS_create_part(address, size, number)`
  - Allocate a dynamic Memory Partition at `address` containing `size` sized `number` of blocks (combination of KS_alloc_part and KS_defpart)
- `KS_free_part(map)`
  - Free the dynamically allocated Memory Partition whose identifier is `map`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug MEMORY PARTITION SNAPSHOT

```
** Partition Snapshot **
  #   Name   Avail/Total Worst Count  Bytes Waiters
  1 PRTSCMAP      3/    4     1     0   2010
  2 AIMAP         0/   20    20   482   64    AINP
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC TRAINING COURSE

PRACTICAL EXERCISES

- **Exercise #2**
  - Defining a Second static task
  - Using Suspend and Resume to synchronize two tasks
- **Exercise #3a**
  - Simulate a crude traffic light
  - Suspend and Resume tasks

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TRAINING

UNIT 4

Semaphores

Timers and Resources

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC SEMAPHORES

EVENT SYNCHRONIZATION OBJECT

- Associated with process or software generated events
- Tri-state model: PENDING, WAITING or DONE
- State automatically maintained by RTXC
- Only one task can wait on a semaphore
- Multiple semaphores can be associated with one event

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

PENDING STATE

- Event is expected but next occurrence has not yet happened
- No task is waiting on the event to occur
- All semaphores set to PENDING state by default during system initialization
- Semaphore state = 0xFF (assuming a character sized definition of type SEMA)

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

DONE STATE

- Expected event occurred but no task was waiting for it
- Transitions from PENDING to DONE as the result of a KS_signal, KS_signalm, KS_ISRsignal, KS_ISRsignalm, or possibly KS_ISRexit
- Semaphore state = 0x00
- Transition from DONE to PENDING occurs as result of KS_wait, KS_waitt or KS_waitm
  - Task does not wait
  - Requesting task retains status as Current Task

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

WAITING STATE

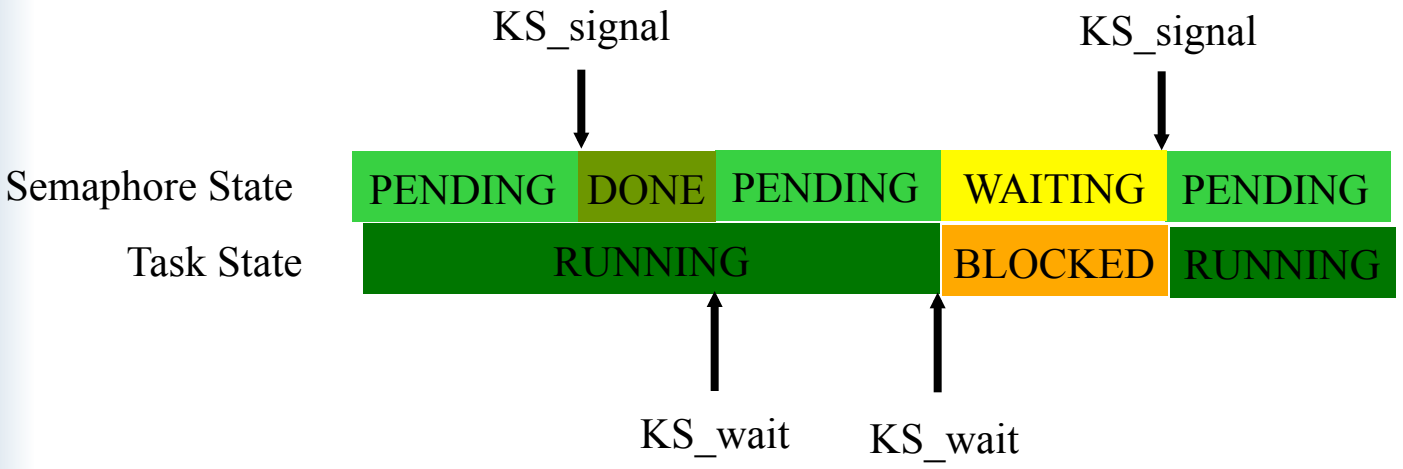
- Event is expected but next occurrence has not yet happened
- Transition from PENDING to WAITING occurs as result of KS_wait, KS_waitt or KS_waitm
  - A task is waiting on the event to occur
  - Task is blocked while waiting
- Semaphore state = 0x01 - 0xFE
  - Content of semaphore is the handle (ID) of the task waiting for the event
- Transition from WAITING to PENDING occurs a result of KS_signal, KS_signalm, KS_ISRsignal, KS_ISRsignalm, or possibly KS_ISRexit

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SEMAPHORE AND TASK COORDINATION

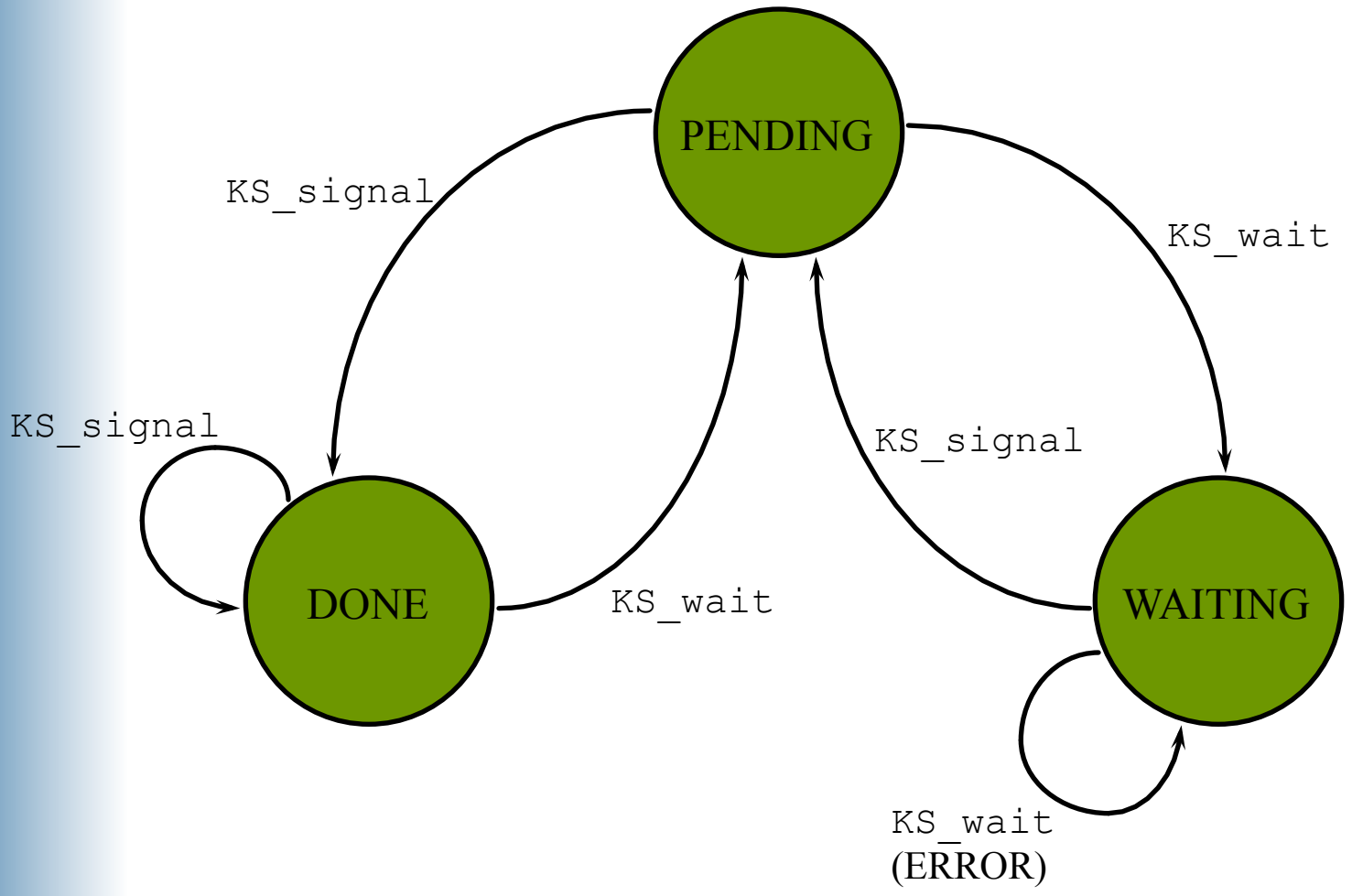
- Events can occur in two ways:
  - Before task attempts to synchronize with event using KS_wait or KS_waitm
  - After task attempts to synchronize with event using KS_wait or KS_waitm



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

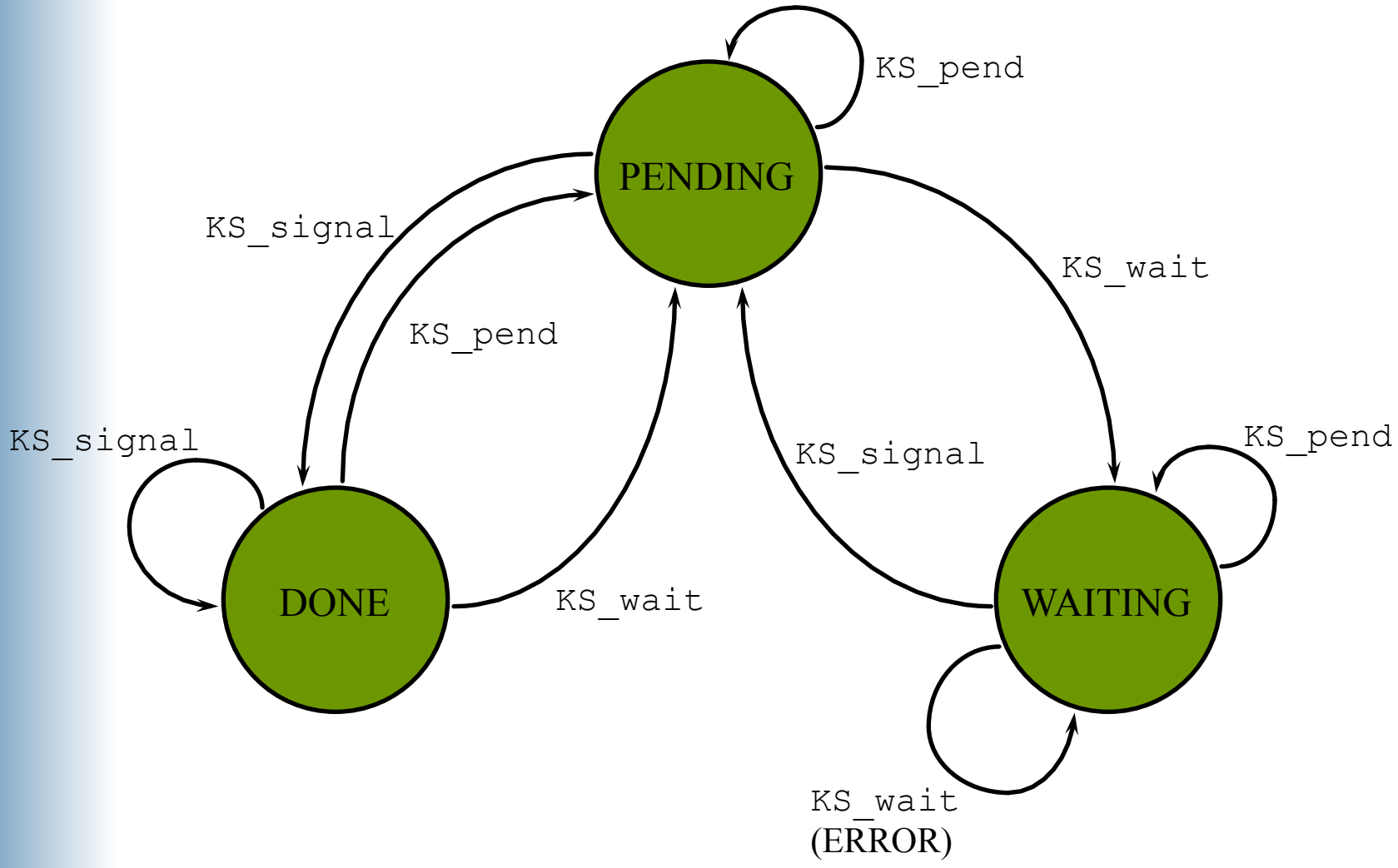
SEMAPHORE STATE TRANSITIONS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

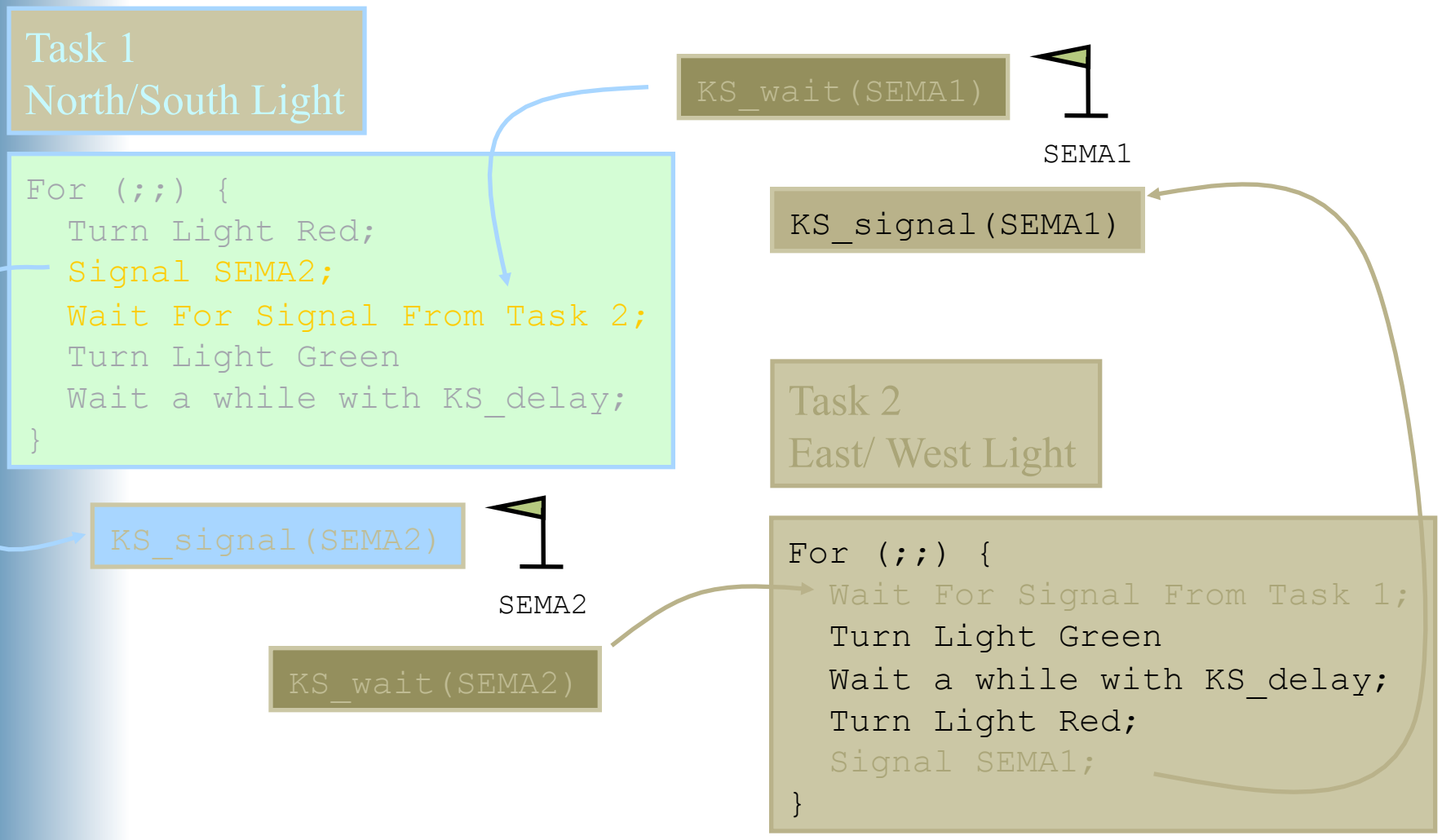
SEMAPHORE STATE TRANSITIONS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

SEMAPHORES AND APPLICATIONS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC SEMAPHORES

SEMAPHORE KERNEL SERVICES

- `KS_wait(semaphore)`
  - Cause task to wait for occurrence of event associated with `semaphore`
- `KS_waitt(semaphore,ticks)`
  - Same as KS_wait( ) but duration of wait limited to a period defined by `ticks`
- `KS_waitm(semaphore_list)`
  - Cause task to wait for occurrence of an event associated with any semaphore found in `semaphore_list` (logical OR condition) and returns the ID of the semaphore associated with the event
- `KS_signal(semaphore)`
  - Signal `semaphore` that associated event has occurred

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC SEMAPHORES

SEMAPHORE KERNEL SERVICES

- `KS_signalm(semaphore_list)`
  - Signal multiple semaphores as specified in `semaphore_list` that a particular event has occurred
- `KS_pend(semaphore)`
  - Set `semaphore` to a `PENDING` state
- `KS_pendm(semaphore_list)`
  - Set multiple semaphores as specified in `semaphore_list` to a `PENDING` state
- `KS_inqsema(semaphore)`
  - Get current state of `semaphore`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug SEMAPHORE SNAPSHOT

```
** Semaphore Snapshot **
  #   Name   State   Waiter
  1 PRNSEMA   PEND
  2 PRTSCSEM DONE
  3 COMISEMA WAIT COMIDRV
  4 DEMOSEM0 WAIT DEMO1 <QNE> DEMOQ
  5 DEMOSEM1 WAIT DEMO1 <MBXNE> DEMOMBOX
  6 DINTSEMA WAIT DINP
  7 SDINSEMA WAIT DINP
```

Semaphore States:

PEND	Pending - Semaphore not yet set
WAIT	Waiting - Tasks waiting for Semaphore
DONE	Done - Semaphore set but no tasks waiting

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TIMERS

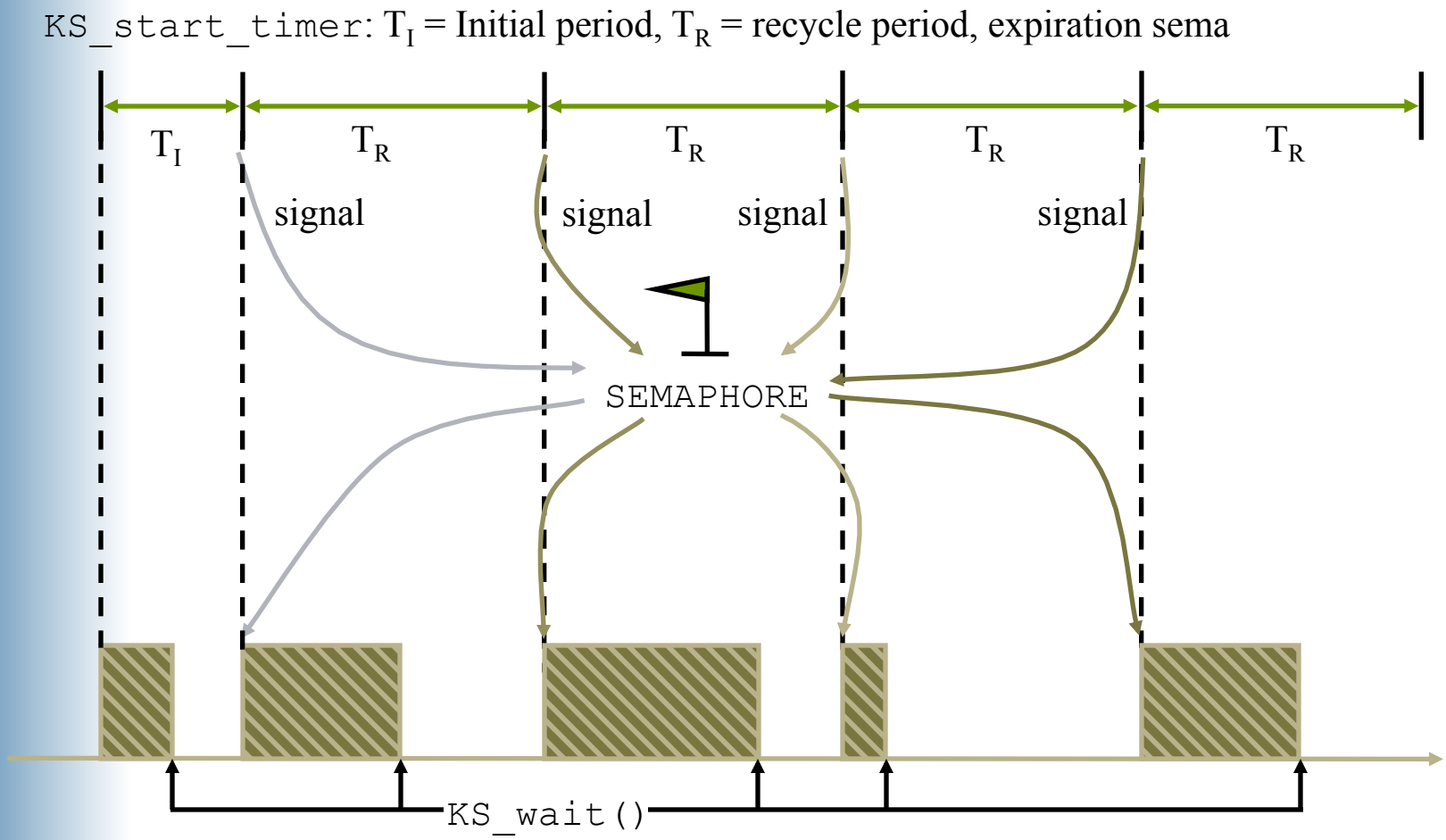
TIME MANAGEMENT OBJECT

- Used for general purpose timing
- One-shot and cyclic timers supported
- More than one timer per task permitted
- Time resolution dependent on system time base

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

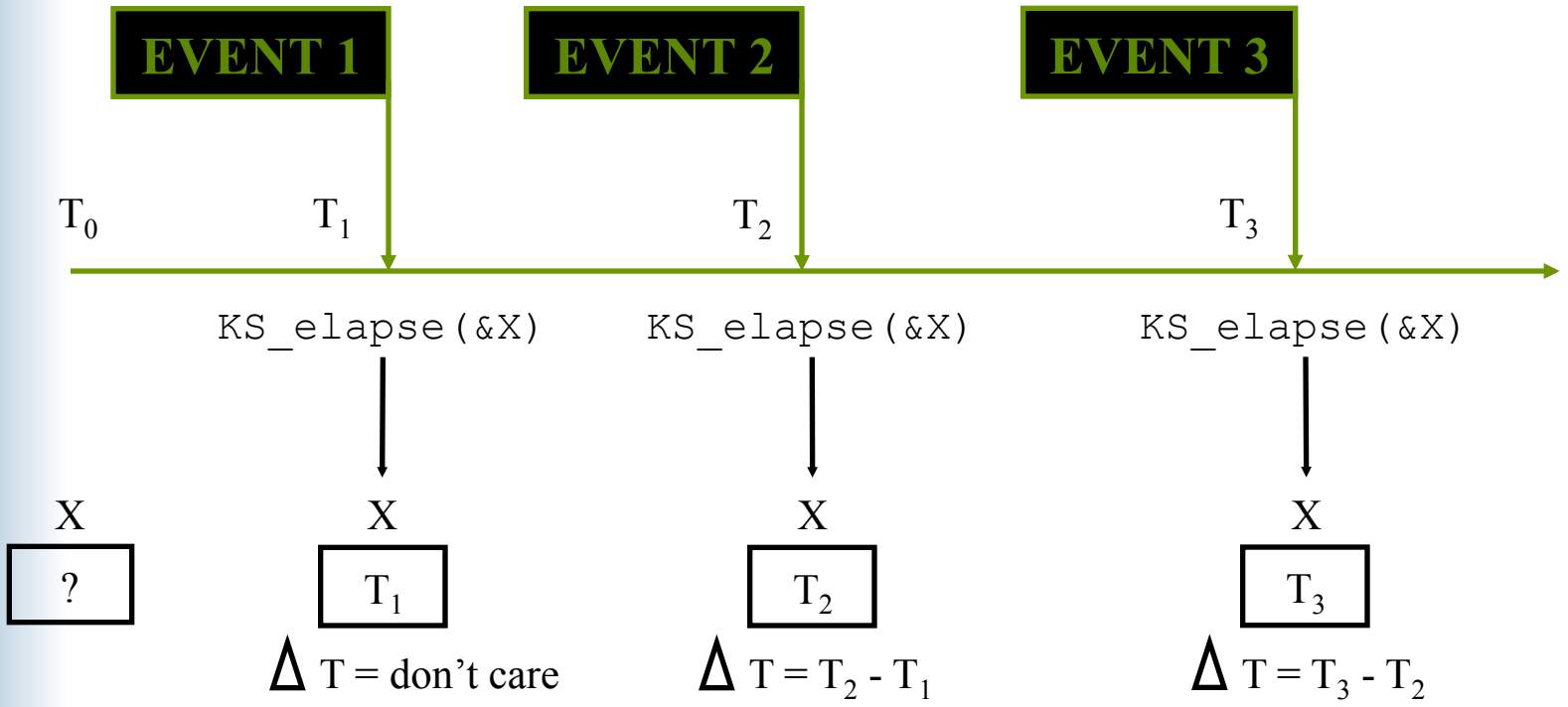
TIMER OPERATION



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

ELAPSED TIME CALCULATION



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TIMERS

TIMER KERNEL SERVICES

- `KS_alloc_timer(void)`
  - Allocate a timer block and return its address
- `KS_start_timer(timer,period,cyclic_period, semaphore)`
  - Start `timer` giving it an initial `period` and an optional `cyclic_period` and associate `semaphore` with the expiration of the initial period or cyclic period
- `KS_restart_timer(timer,period,cyclic_period)`
  - Reset and restart the active `timer` with a new initial `period` and an optional `cyclic_period`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC TIMERS

TIMER KERNEL SERVICES

- `KS_stop_timer(timer)`
  - Stop the specified active `timer` and remove it from the active timer list
- `KS_free_timer(timer)`
  - Free the `timer` block
- `KS_elapse(counter)`
  - Set current time into `counter`, then subsequent call calculates elapsed time
- `KS_inqtimer(timer)`
  - Get the time remaining on an active `timer`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXCbug CLOCK SNAPSHOT

```
** Clock Snapshot **

Clock rate is  100 Hz, Tick interval is 10 ms, Maximum of 16 timers
Tick timer is    37046, ET is      126 ticks, RTC time is       370

   Time       Cyclic   Task    Timer  Object
 Remaining     Value   Name    Type    Name
       500      1000   CAL     Timer  CALSEMA
      7500         0   TACH1   Delay  TACH1
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC RESOURCES

EXCLUSIVE ACCESS OBJECT

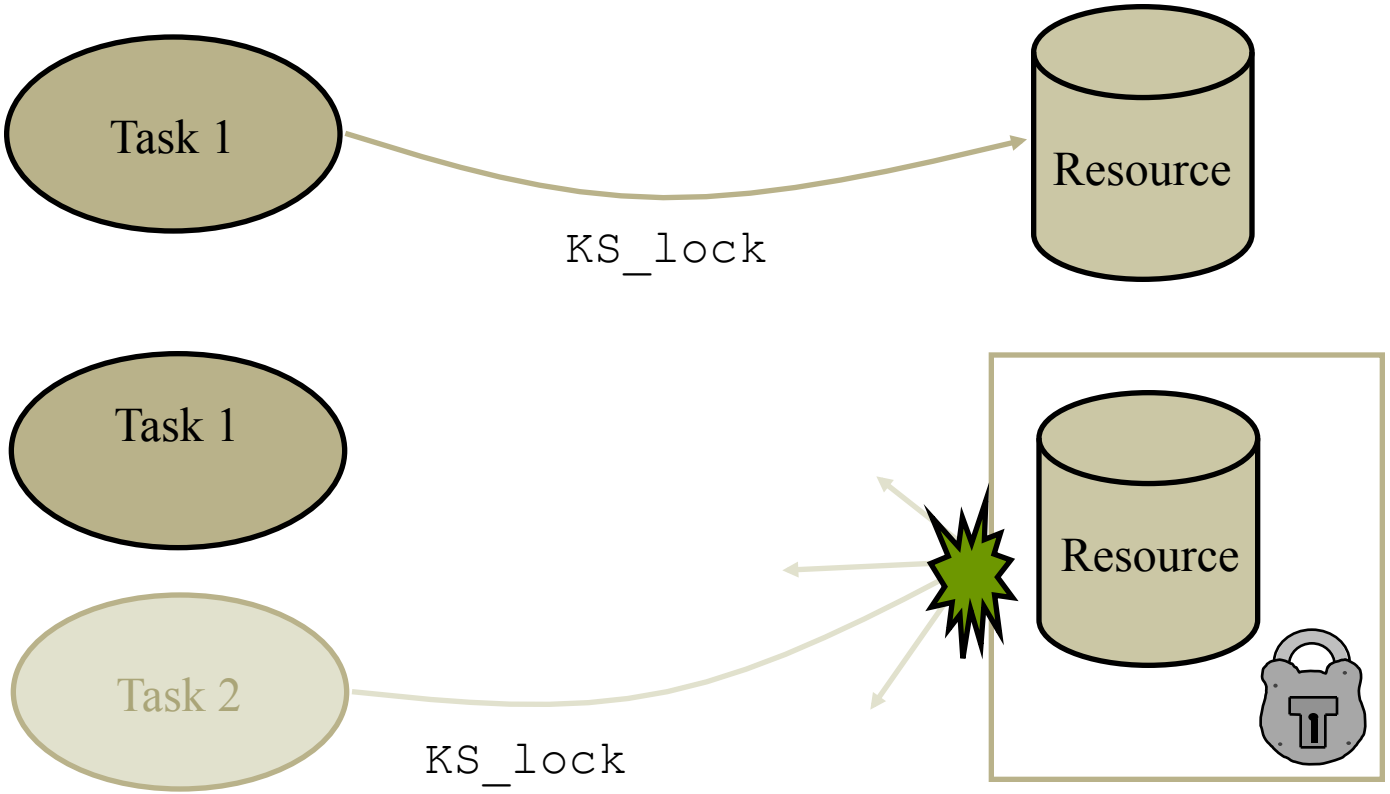
- Resource may be any entity: Hardware or Software
- Resource permits exclusive access to entity by owner
- One owner at a time
- Priority inversion optionally handled by RTXC
- RTXC manages tasks waiting to become resource owner
- Caution: Nested ownership is allowed, make sure you match `KS_locks` and `KS_unlocks`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC RESOURCES

RESOURCE MANAGEMENT

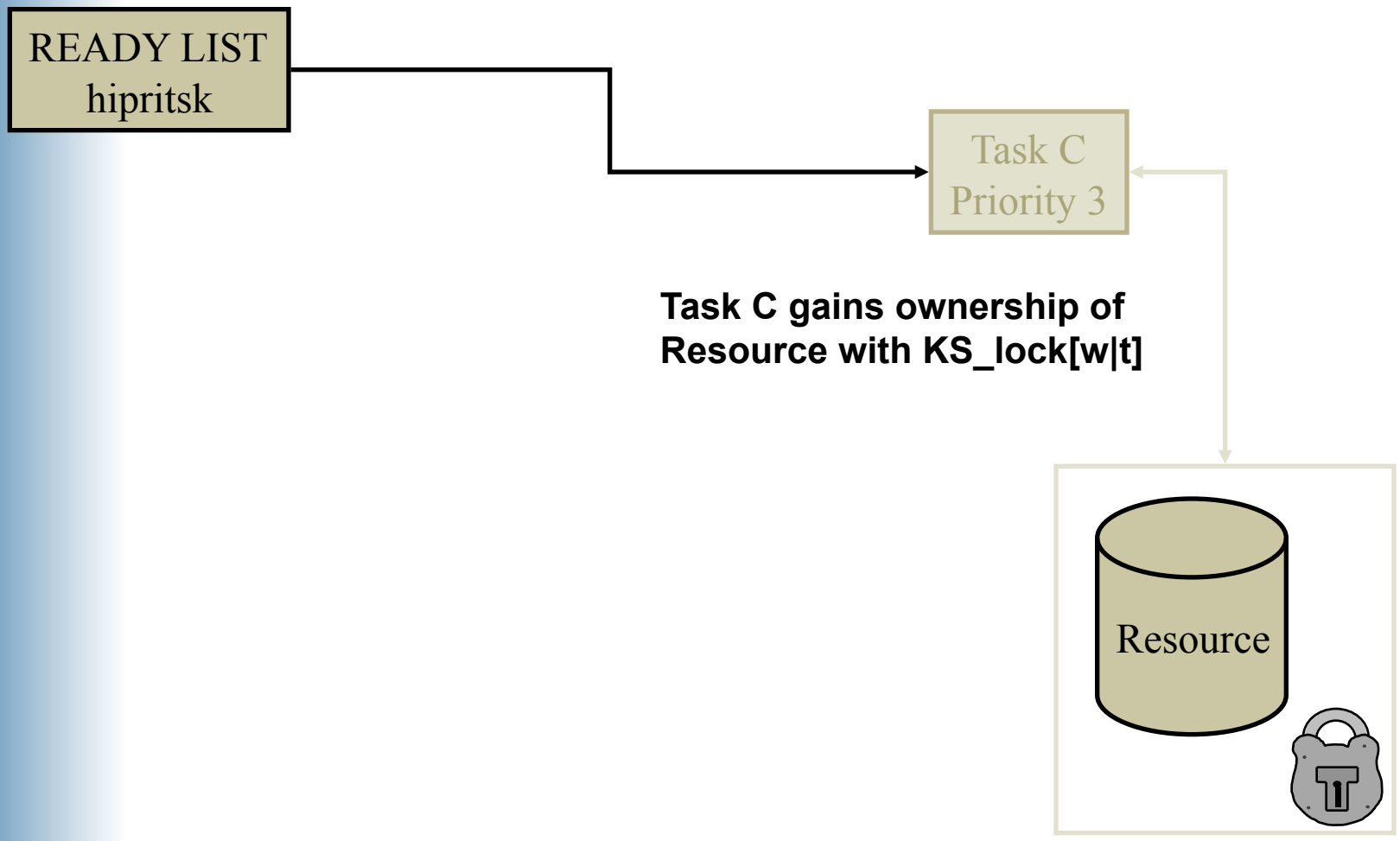


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC RESOURCES

PRIORITY INVERSION

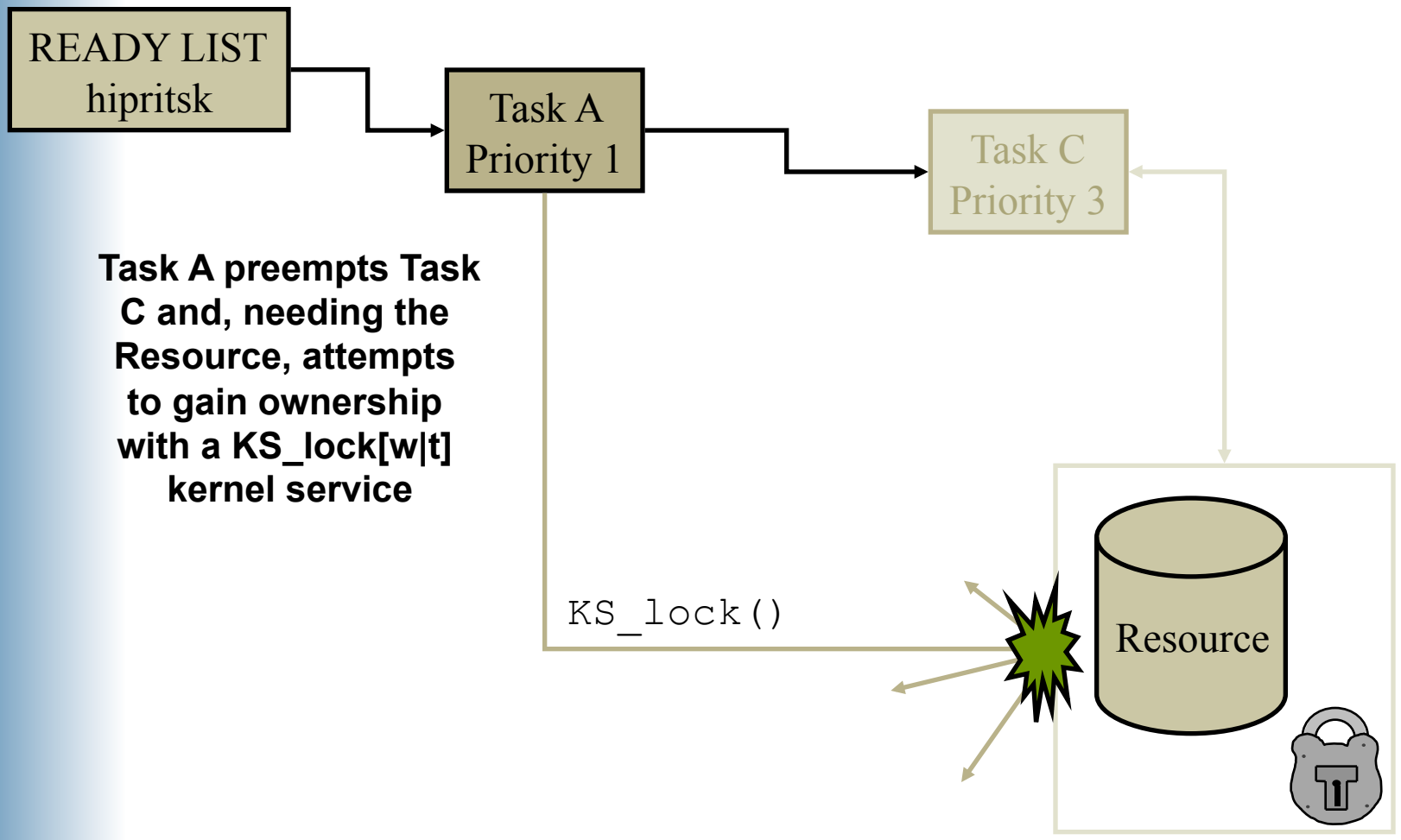


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC RESOURCES

PRIORITY INVERSION

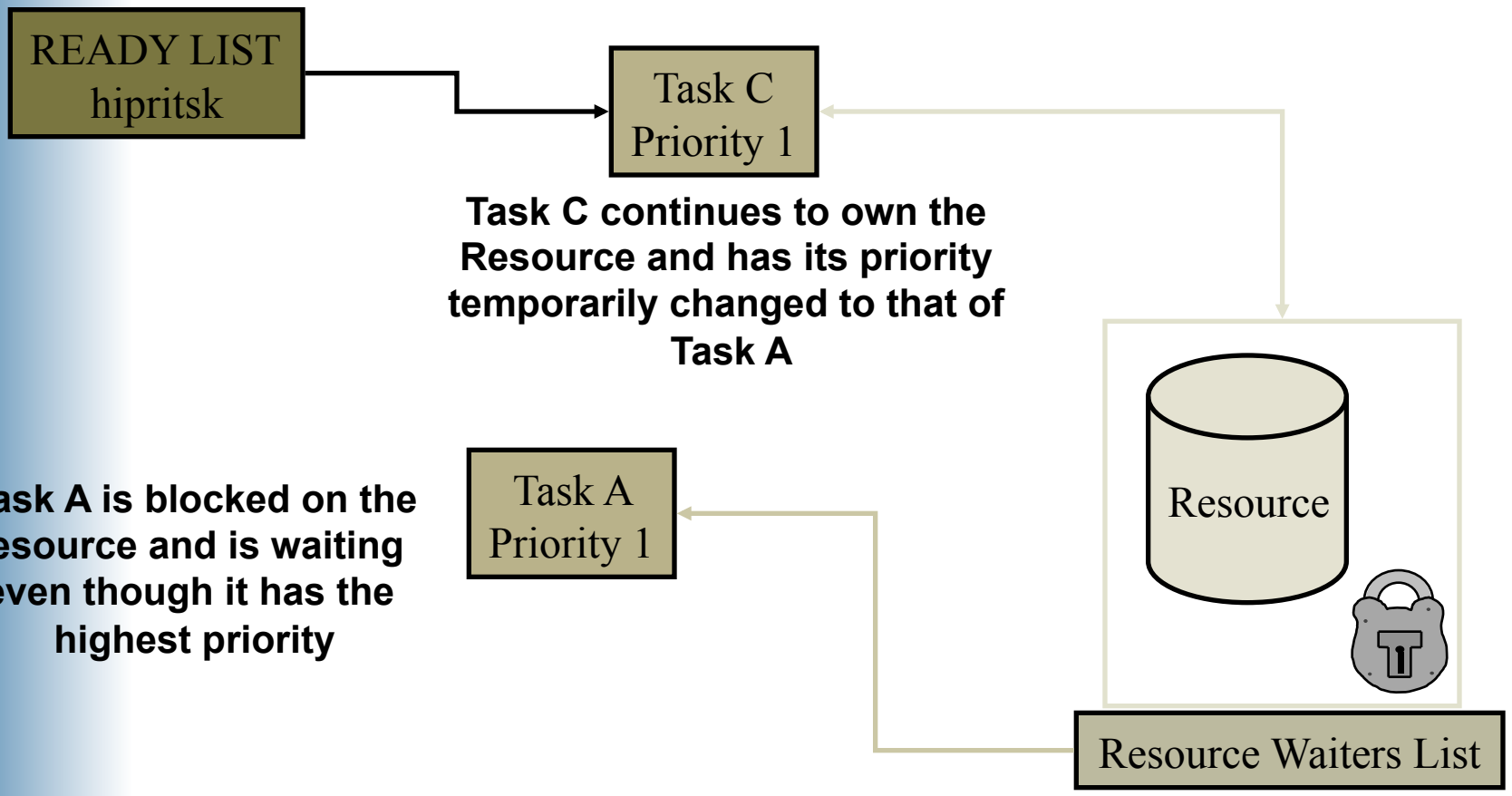


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC RESOURCES

PRIORITY INHERITANCE

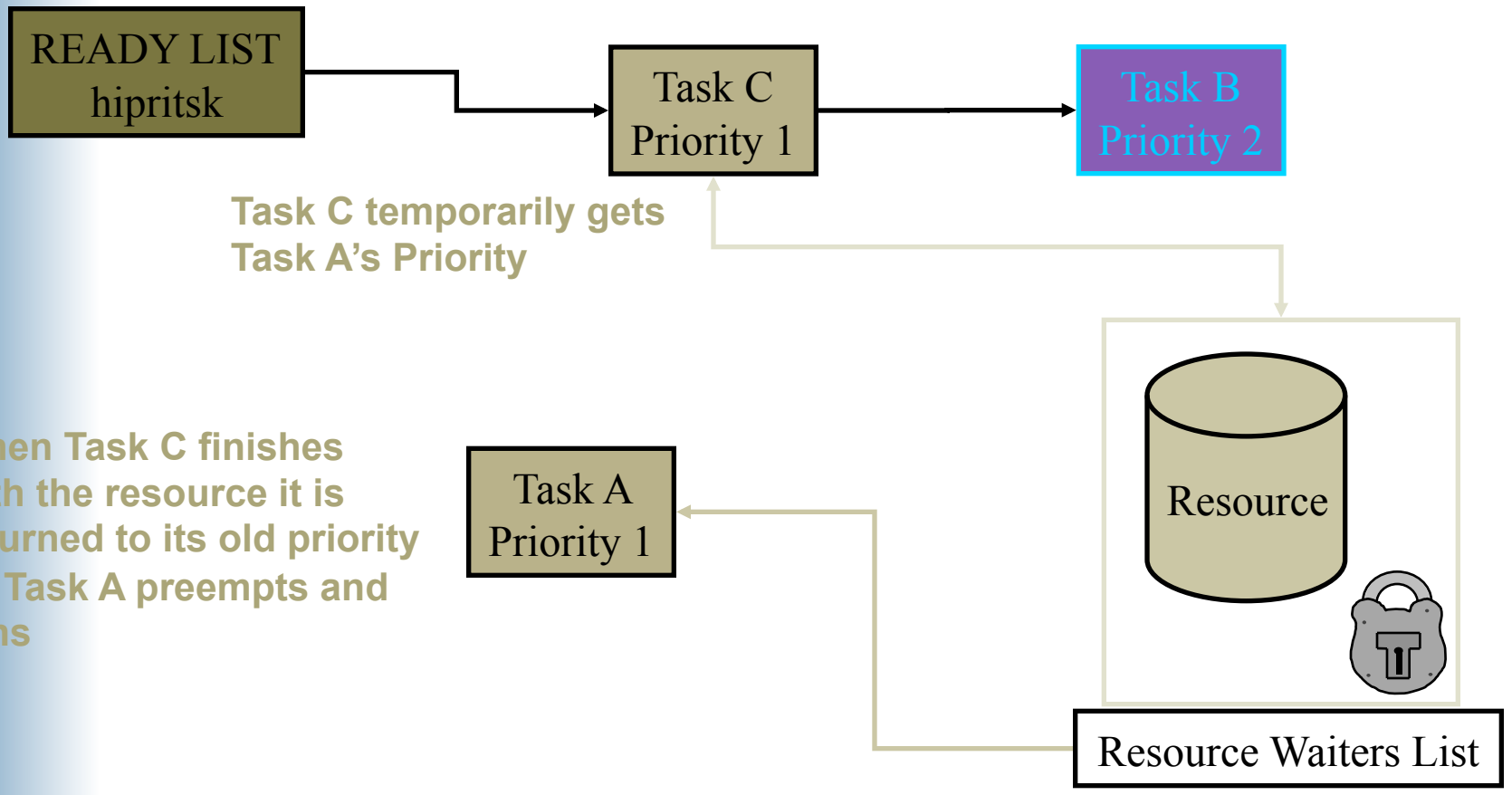


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC RESOURCES

PRIORITY INHERITANCE



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC RESOURCES

RESOURCE KERNEL SERVICES

- `KS_lock(resource)`
  - Acquire exclusive use of `resource`, returns `RC_GOOD` if successful, `RC_BUSY` if resource locked by another owner
- `KS_lockw(resource)`
  - Same as `KS_lock()` but, if `resource` is `BUSY`, wait until `resource` available
- `KS_lockt(resource,ticks)`
  - Same as `KS_lockw()` but limit duration of wait to period of `ticks`
- `KS_unlock(resource)`
  - Release exclusive ownership of `resource`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC RESOURCES

RESOURCE KERNEL SERVICES

- `KS_defres(resource, resattr)`
  - Enable or disable the priority inversion attribute for the `resource`
- `KS_inqres(resource)`
  - Get the identifier of the owner of a `resource`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug RESOURCE SNAPSHOT

```
** Resource Snapshot **
  #    Name    Count Conflicts    Owner    Waiters
  1 PRNRES    26742         1
  2 DOSRES        1         2   PRTSCRN     FILMGR
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TRAINING COURSE

PRACTICAL EXERCISES

- **Exercise #5a**
  - Using Semaphores to Synchronize Tasks
- **Exercise #6a**
  - Using Cyclic timers with semaphores

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TRAINING

UNIT 5

Queues

Mailboxes and Messages

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC FIFO QUEUES

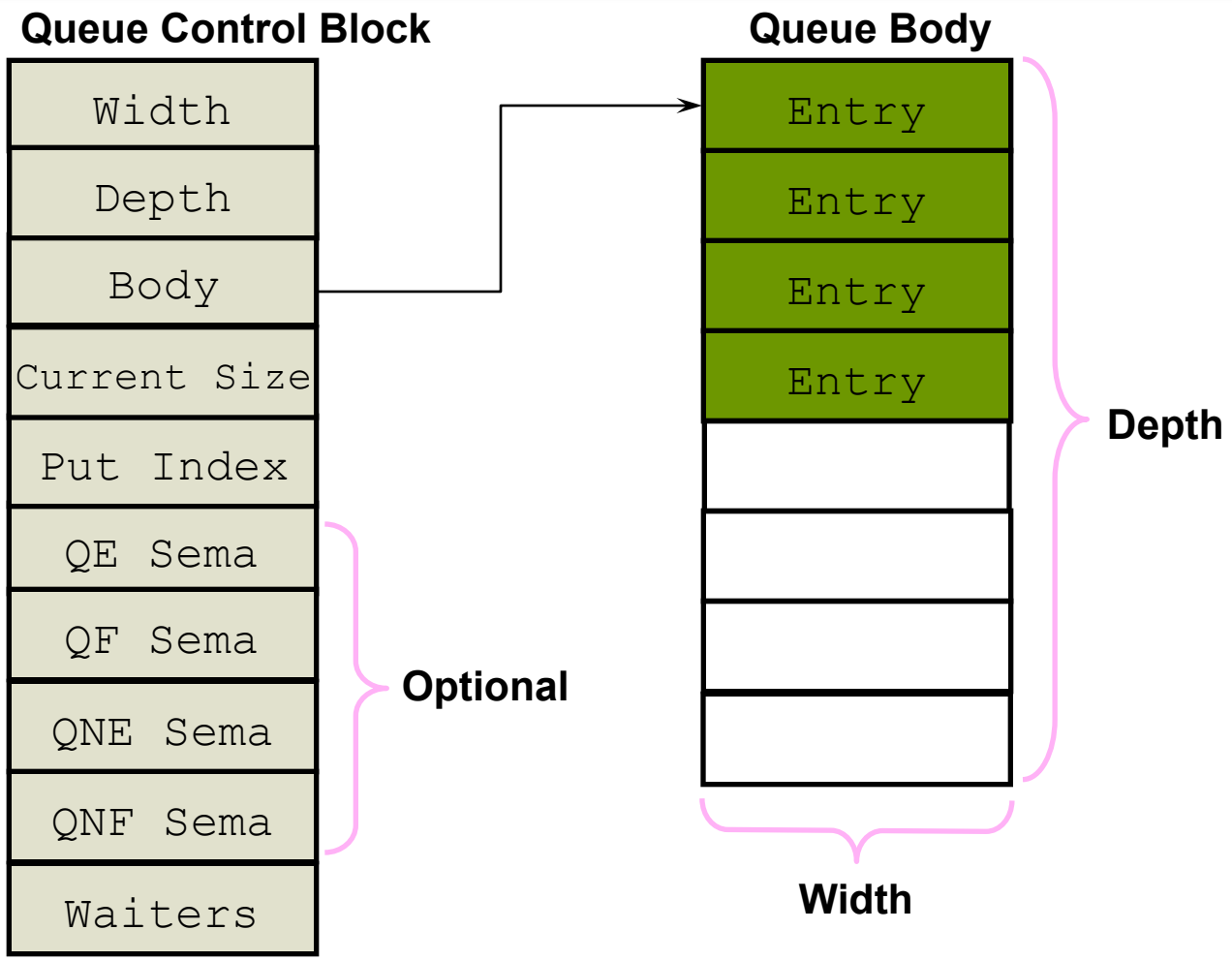
CHRONOLOGICAL DATA MOVEMENT OBJECT

- Supports multiple producers and multiple consumers
- Automatic synchronization possible on EMPTY and FULL conditions when using KS_enqueuew and KS_dequeuew
- Can be used with queue semaphores when trying to synchronize with multiple events using KS_waitm
  - Not used when using KS_enqueuew or KS_dequeuew
  - Use with multiple queues or with mailboxes or normal events
- Predefined WIDTH and DEPTH
  - All entries in a queue have same Width
  - Different queues can have different Widths and Depths

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

FIFO QUEUE ORGANIZATION

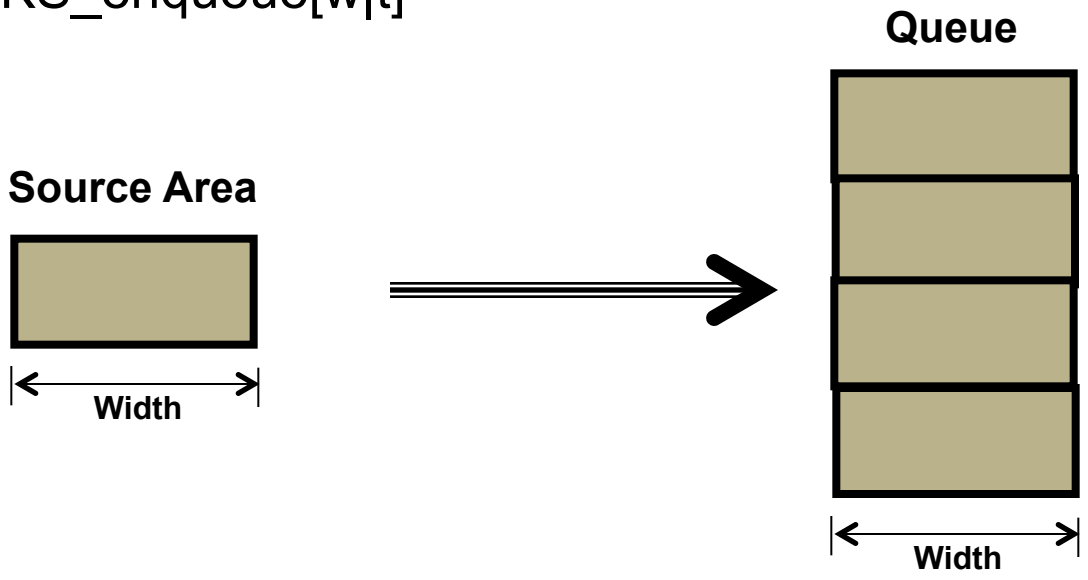


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

PUTTING DATA INTO A QUEUE

- Use a variant of KS_enqueue to put data into a queue
  - KS_enqueuew or KS_enqueuet
- Move data from a source area to the queue
  - Source area should be at least size of queue's Width
  - Give address of source area (&source) as parameter to KS_enqueue[w|t]

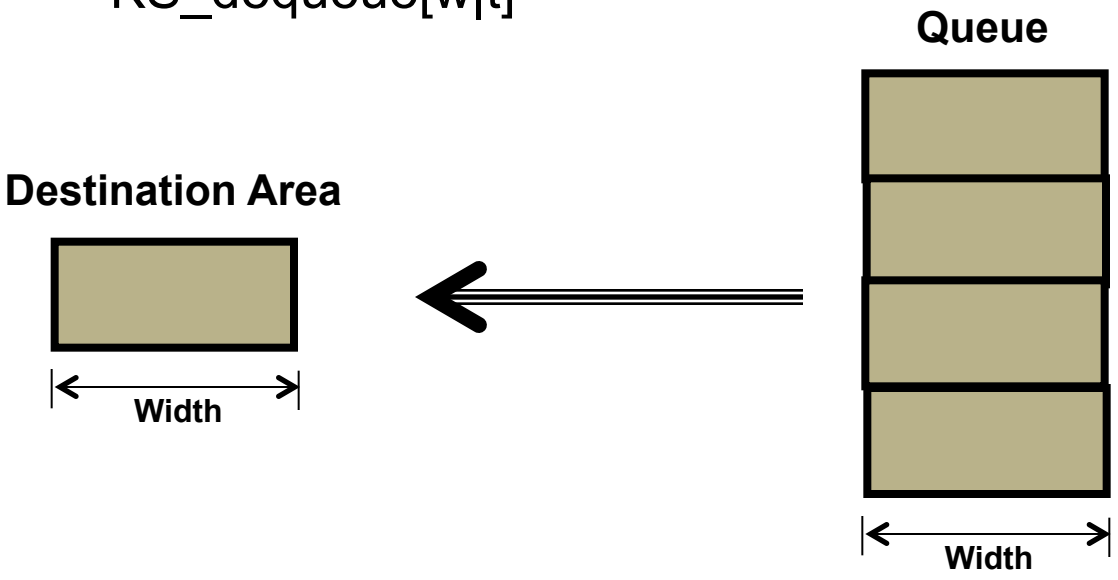


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

GETTING DATA FROM A QUEUE

- Use a variant of KS_dequeue to get data from a queue
  - KS_dequeuew or KS_dequeuet
- Move data from queue to a destination area
  - Destination area should be at least size of queue's Width
  - Give address of destination area (&dest) as parameter to KS_dequeue[w|t]

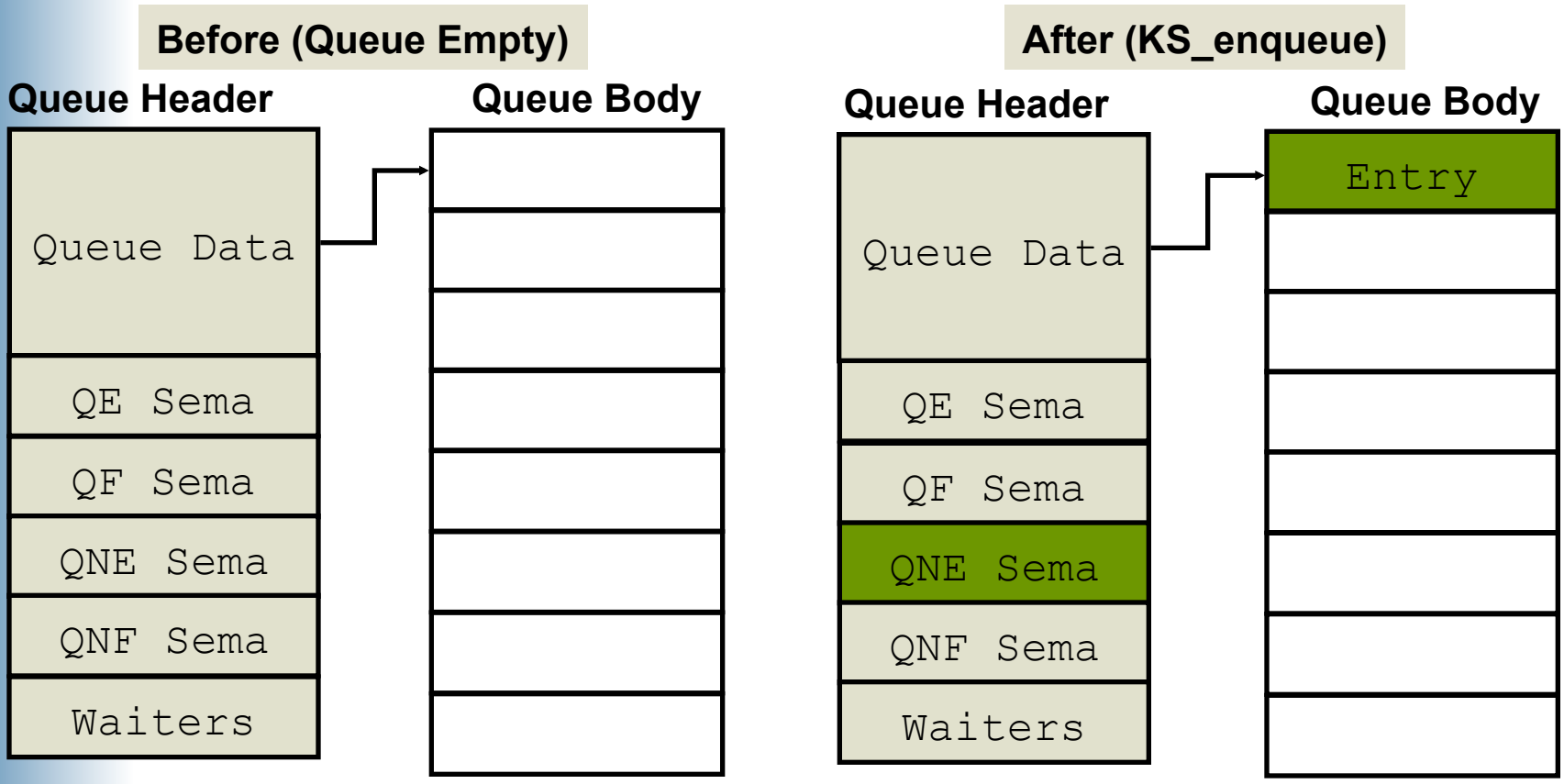


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC QUEUE SEMAPHORE USAGE

Queue Not Empty EVENT

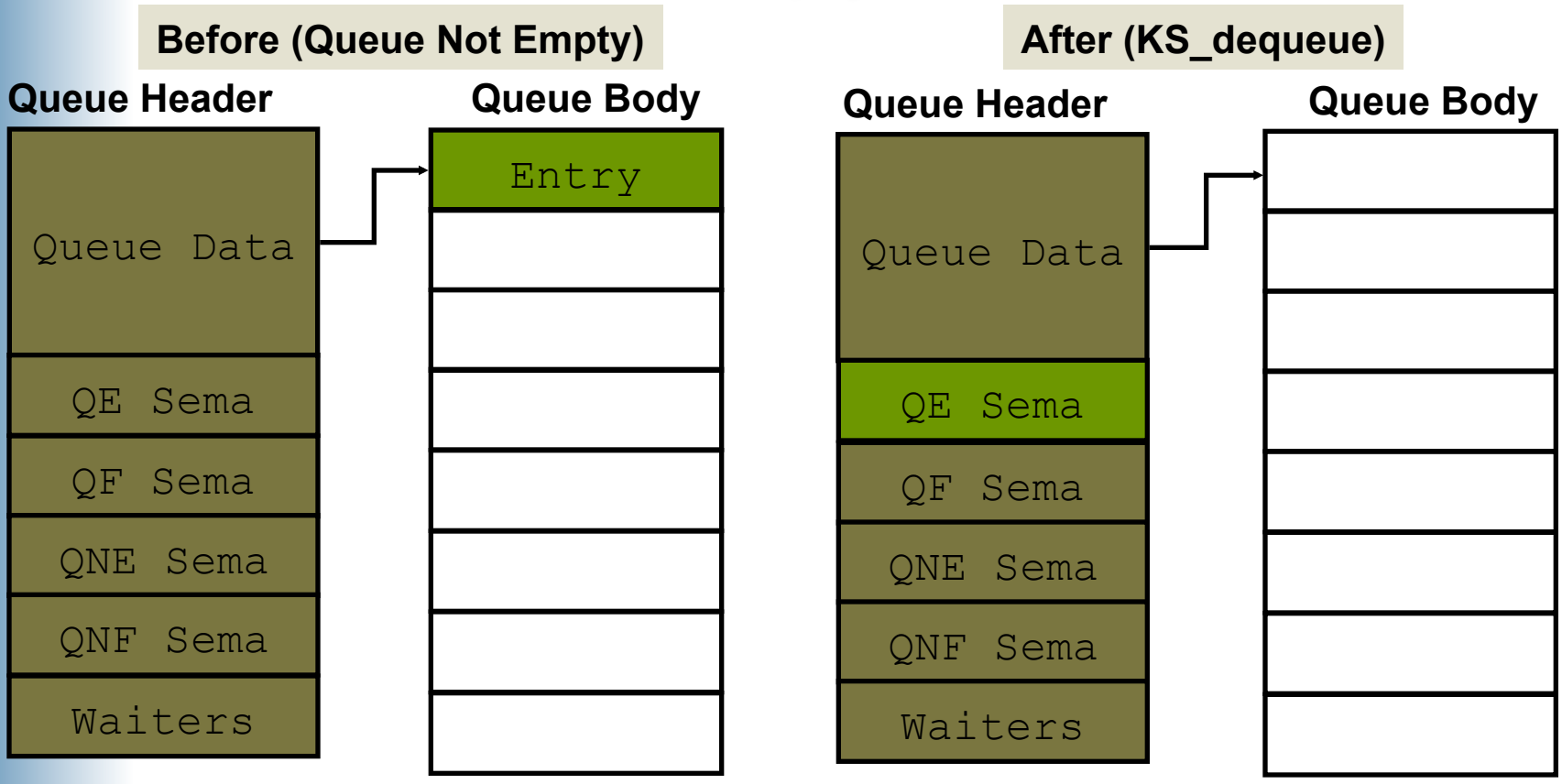


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC QUEUE SEMAPHORE USAGE

Queue Empty EVENT

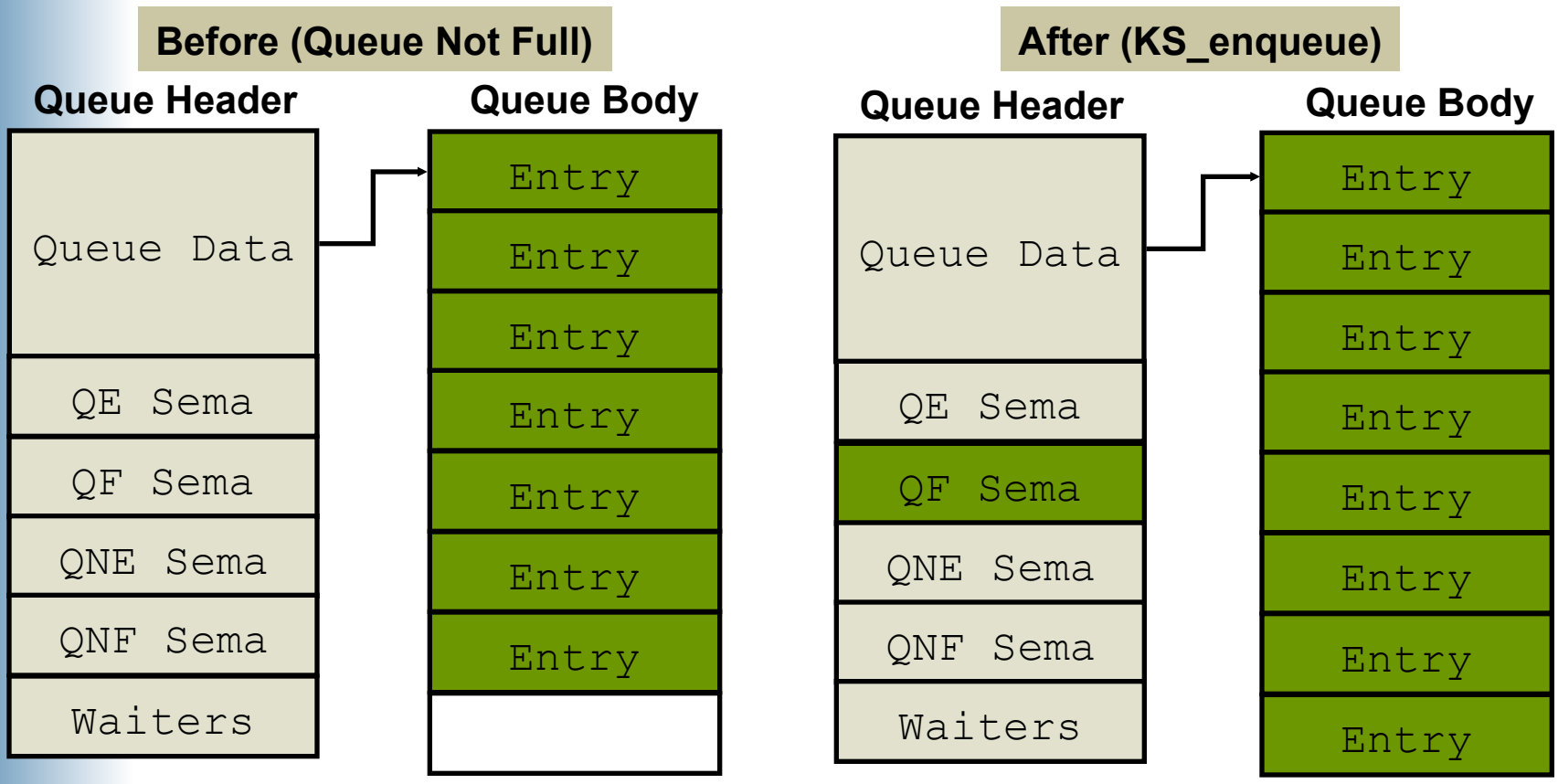


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC QUEUE SEMAPHORE USAGE

Queue Full EVENT

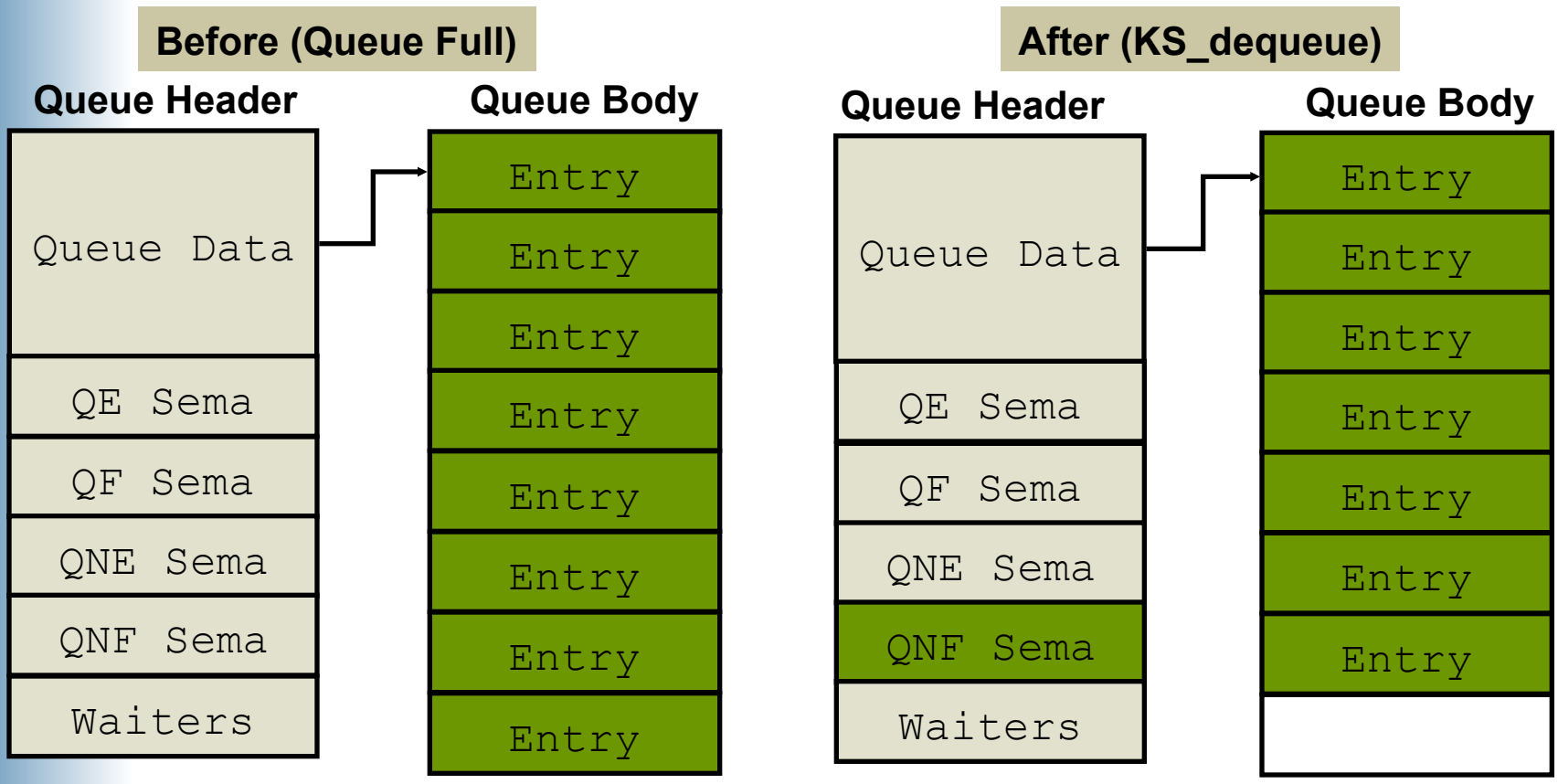


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC QUEUE SEMAPHORE USAGE

Queue Not Full EVENT



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC QUEUES

QUEUE KERNEL SEVICES

- `KS_dequeue(queue,destination)`
  - Get data from `queue` and store it at `destination` address
- `KS_dequeuew(queue,destination)`
  - Same as `KS_dequeue()` but, if `queue` is `EMPTY`, wait until operation can be completed
- `KS_dequeuet(queue,destination,ticks)`
  - Same as `KS_dequeuew()` but duration of wait is limited by period defined by `ticks`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC QUEUES

QUEUE KERNEL SEVICES

- `KS_enqueue(queue,source)`
  - Put data at `source` address into `queue`
- `KS_enqueuew(queue,source)`
  - Same as `KS_enqueue()` but, if `queue` is `FULL`, wait until there is room in queue to complete the operation
- `KS_enqueuet(queue,source,ticks)`
  - Same as `KS_enqueuew()` but limit duration of wait to period defined by `ticks`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC QUEUES

QUEUE KERNEL SEVICES

- `KS_defqsema(queue,semaphore,condition)`
  - Associate `semaphore` with the given `condition` on `queue`
- `KS_defqueue(queue,width,depth,body,count)`
  - Define `width`, `depth`, queue `body` address and current size of `queue`
- `KS_inqqueue(queue)`
  - Get the number of entries currently in `queue`
- `KS_purgequeue(queue)`
  - Purge `queue` of all entries

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXCbug QUEUE SNAPSHOT

```
** Queue Snapshot **
  #   Name  Current/Depth Worst Count Waiters
  1 CONIQ          0/   16     1    19
  2 CONOQ        108/ 1024   546  3413
  3 COMOQ          0/  128     0     0 COMODRV
  4 DEMOQ          0/   64    40   420 DEMOSEM0 <NE> DEMO1
```

Queue Semaphore Condition Codes:

<E>	Empty
<F>	Full
<NE>	Not Empty
<NF>	Not Full

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC KERNEL OBJECTS

- TASKS for application functions
- MEMORY PARTITIONS for RAM memory management
- SEMAPHORES for event synchronization
- TIMERS for timed operations
- RESOURCES for exclusive access
- QUEUES for chronological data passing
- MAILBOXES and MESSAGES for message transmission

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC MAILBOXES

MESSAGE INTERFACE BETWEEN TWO TASKS

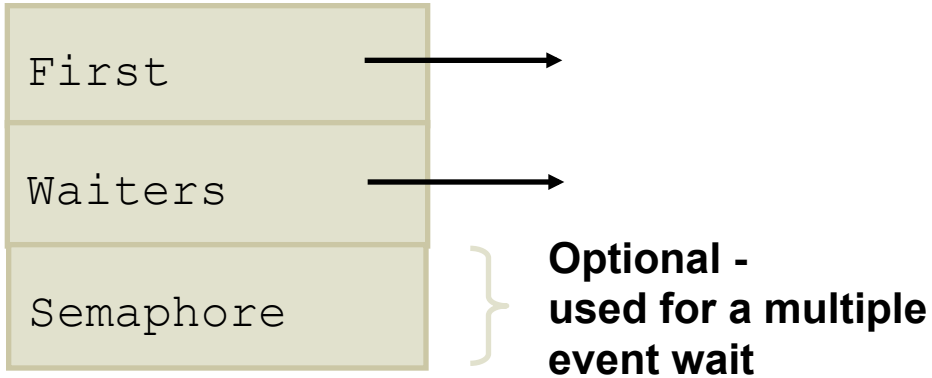
- Mailboxes are defined globally
- Any task can send mail to any mailbox
- A task may use none, one or many mailboxes
- Good design usually dictates one consumer

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC MAILBOXES

MAILBOX STRUCTURE

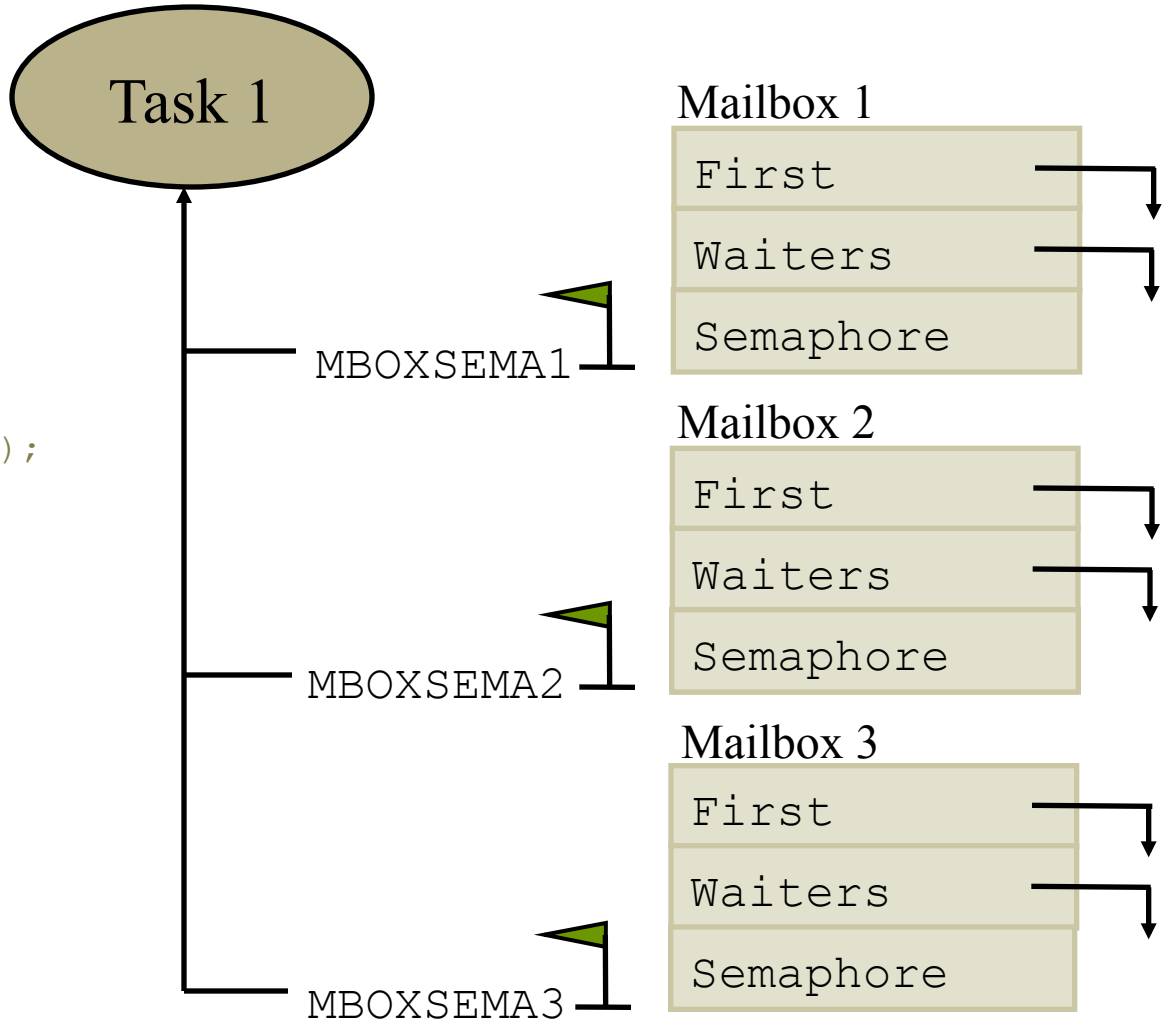


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

USING SEMAPHORES TO SERVICE MULTIPLE OBJECTS

```
SEMA cause;
SEMA semalist[] = {
   MBOXSEMA1,
   MBOXSEMA2,
   MBOXSEMA3,
   0
}

for (;;) {
   cause = KS_waitm(semalist);
   switch(cause) {
         case MBOXSEMA1:
            ...; break;
         case MBOXSEMA2:
            ...; break;
         case MBOXSEMA3:
            ...; break;
   }
}
```



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC MESSAGES

PRIORITY DATA MOVEMENT OBJECT

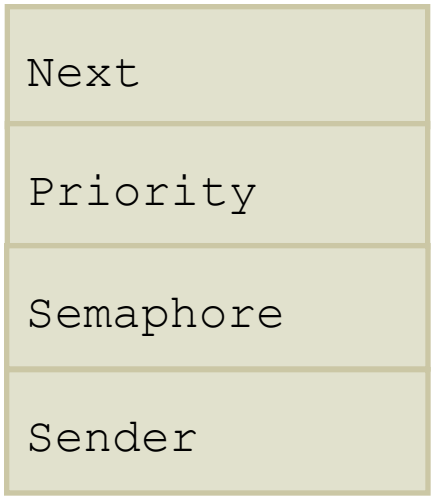
- Data is sent to a receiver via a mailbox
- Messages are inserted according to priority or FIFO if they have the same priority
- Messages are generally processed in the order of their priorities but a task can receive from a specified task if desired
- No data is moved, only pointers
- Body of message is defined by user
- Message body can contain a response message from receiver

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC MESSAGES

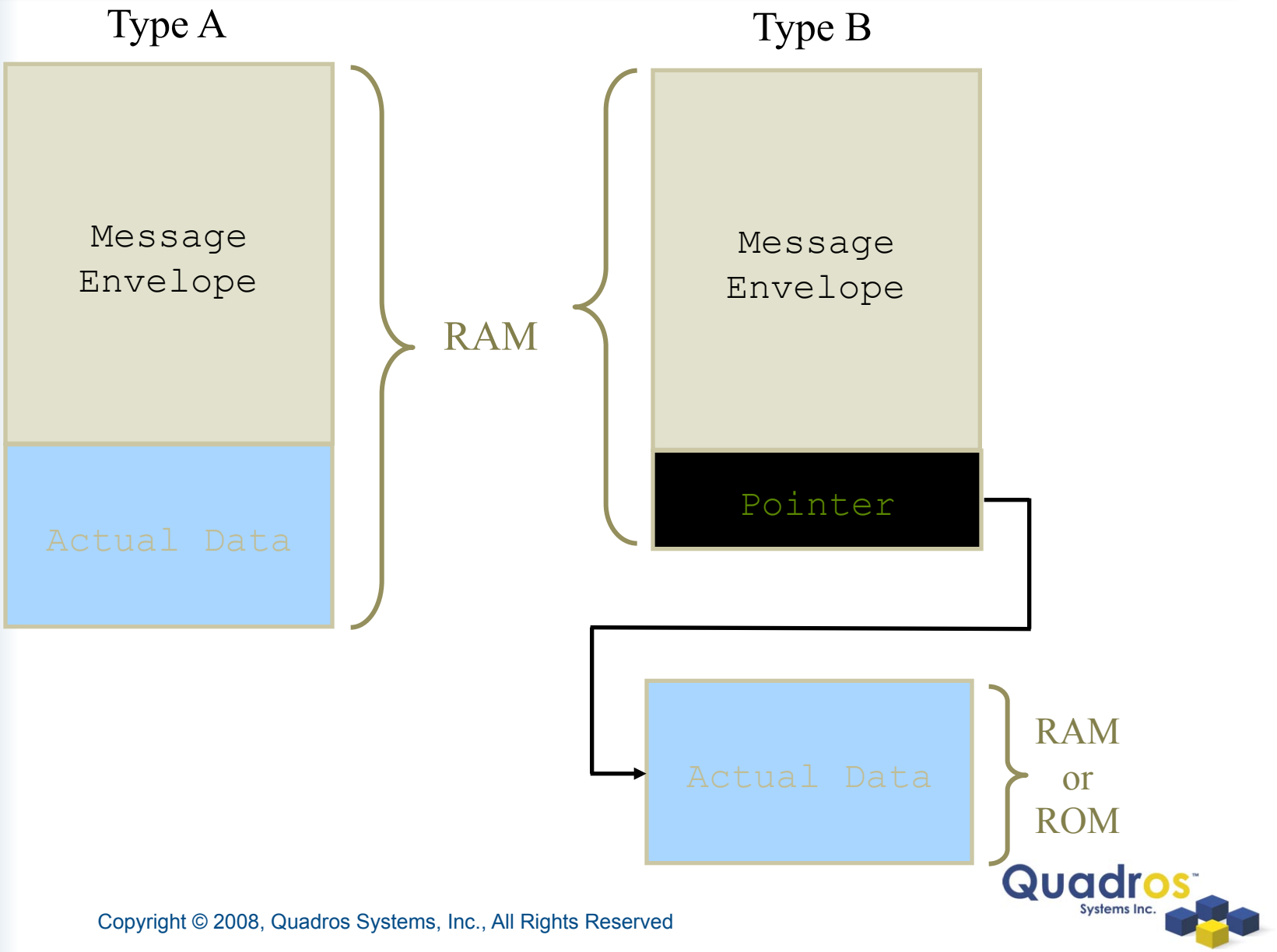
ENVELOPE



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

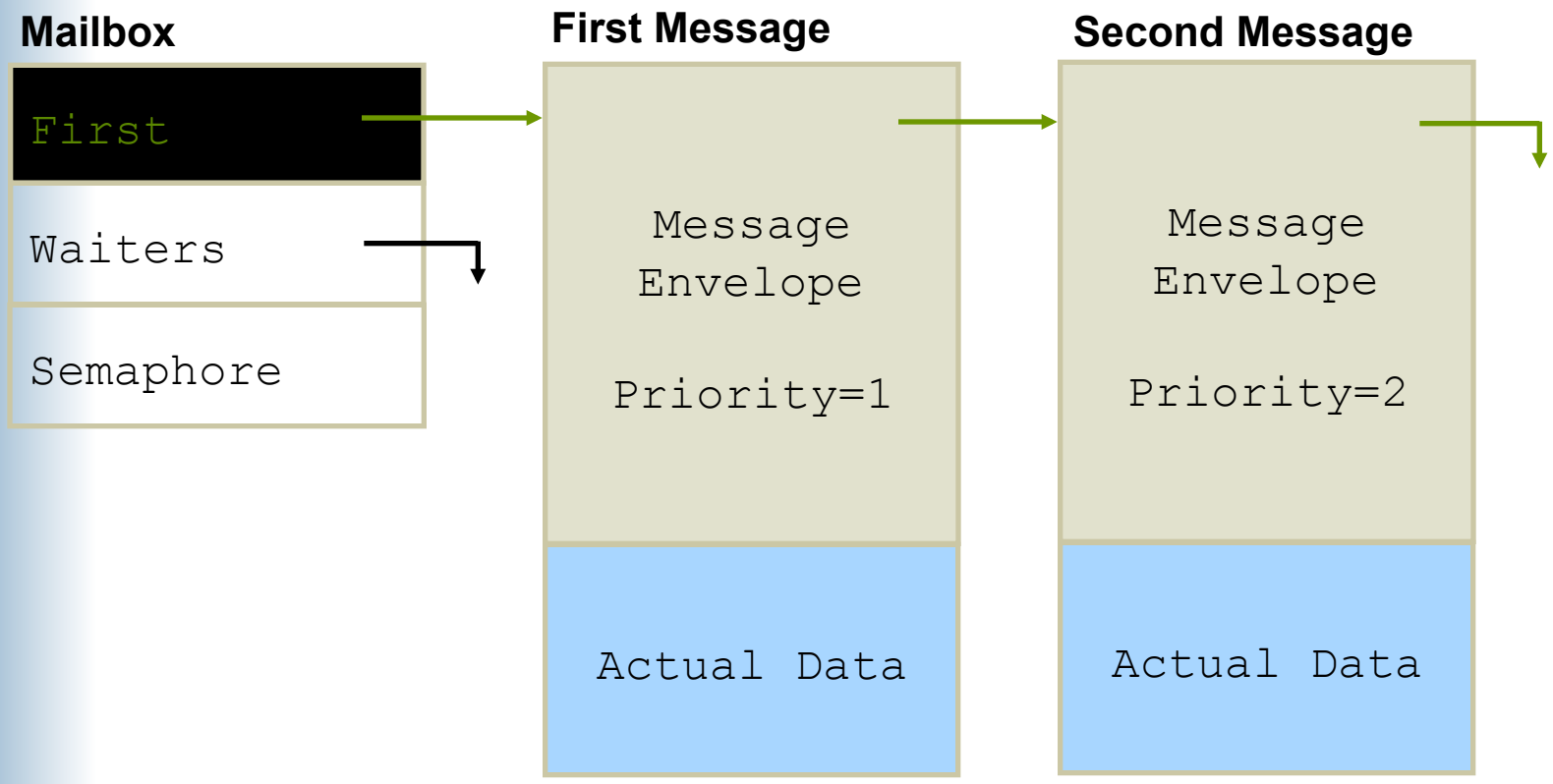
RTXC MESSAGES



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

RTXC MESSAGES

MAILBOX WITH MESSAGES

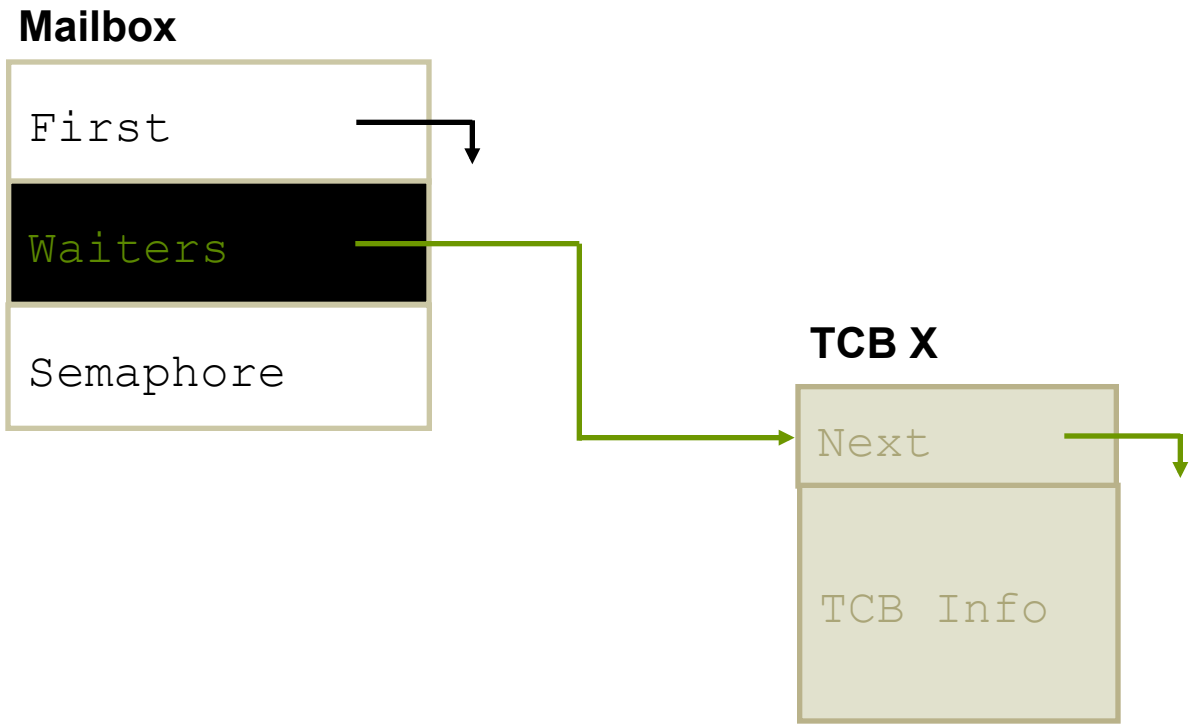


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC MESSAGES

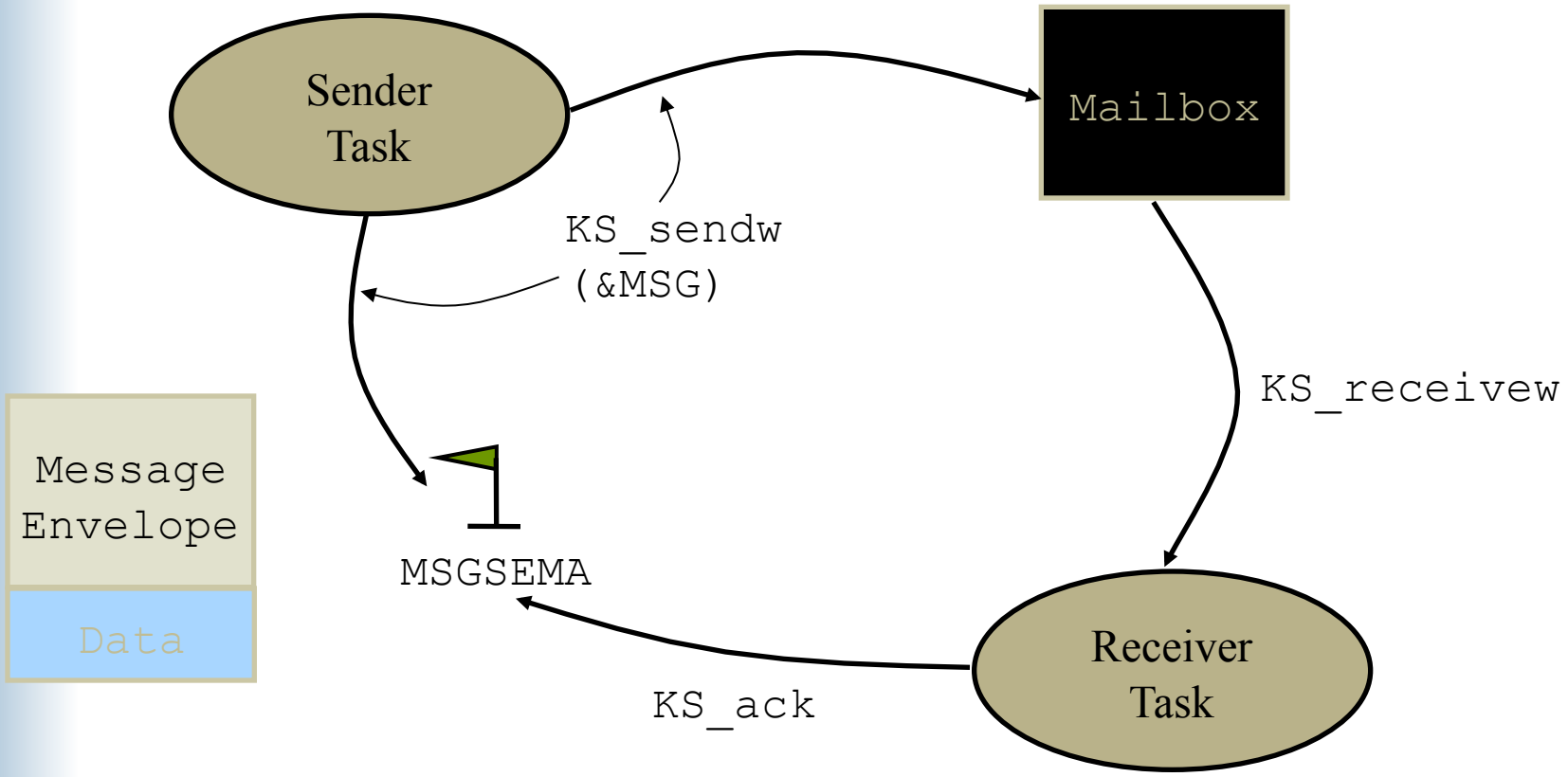
MAILBOX WITH WAITERS



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

MESSAGE PASSING



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC MAILBOXES AND MESSAGES

MESSAGE KERNEL SERVICES

- `KS_receive(mailbox,task)`
  - Receive next message from any sender (or from a specific sender defined by `task`) in `mailbox` and return message address, if mailbox is empty return NULL
- `KS_receivew(mailbox,task)`
  - Same as `KS_receive()` but, if `mailbox` empty, wait until a message is sent
- `KS_receivet(mailbox,task,ticks)`
  - Same as `KS_receivew()` but period of waiting is limited by duration `ticks`
- `KS_ack(message)`
  - Acknowledge receipt and processing of `message`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC MAILBOXES AND MESSAGES

MESSAGE KERNEL SERVICES

- `KS_send(mailbox,message,priority,semaphore)`
  - Send `message` asynchronously at specified `priority` to `mailbox` and associate `semaphore` with acknowledgement signal
- `KS_sendw(mailbox,message,priority,semaphore)`
  - Send `message` synchronously, same as `KS_send()` but wait on `semaphore` for acknowledgement
- `KS_sendt(mailbox,message,priority,semaphore,ticks)`
  - Same as `KS_sendw()` but waiting period is limited by duration `ticks`
- `KS_defmboxsema(mailbox,semaphore)`
  - Associate `semaphore` with the `Not_Empty` condition of `mailbox`

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXCbug MAILBOX SNAPSHOT

```
** Mailbox Snapshot **
  #    Name   Current Count Waiters
  1 FSRVMBOX      0   31472 FILESRVR
  2 PRNMBOX       0    3720 PRNDRV
  3 DEMOMBOX      0     450
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC
SPECIAL KERNEL SERVICES

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC SPECIAL DIRECTIVES

- `KS_deftime(systime)`
  - Define the current date and time in elapsed seconds from base `systime` (January 1, 1970)
- `KS_inqtime(void)`
  - Get the current date and time expressed as elapsed seconds from base system time (January 1, 1970)
- `KS_nop(void)`
  - Test service, no operation
- `KS_user(function,argument_list)`
  - User defined `function` is called by the standard RTXC KS protocol and `argument_list` passed to it

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC TRAINING COURSE

PRACTICAL EXERCISES

- **Exercise #7**
  - Passing information between tasks using Queues
- **Exercise #8**
  - Use Mailboxes and duplicate the functionality of exercise #7
- **Exercise #9**
  - Create an application that will wait on multiple events
  - Use RTXCbug to view the current state of the application

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC TRAINING

UNIT 6

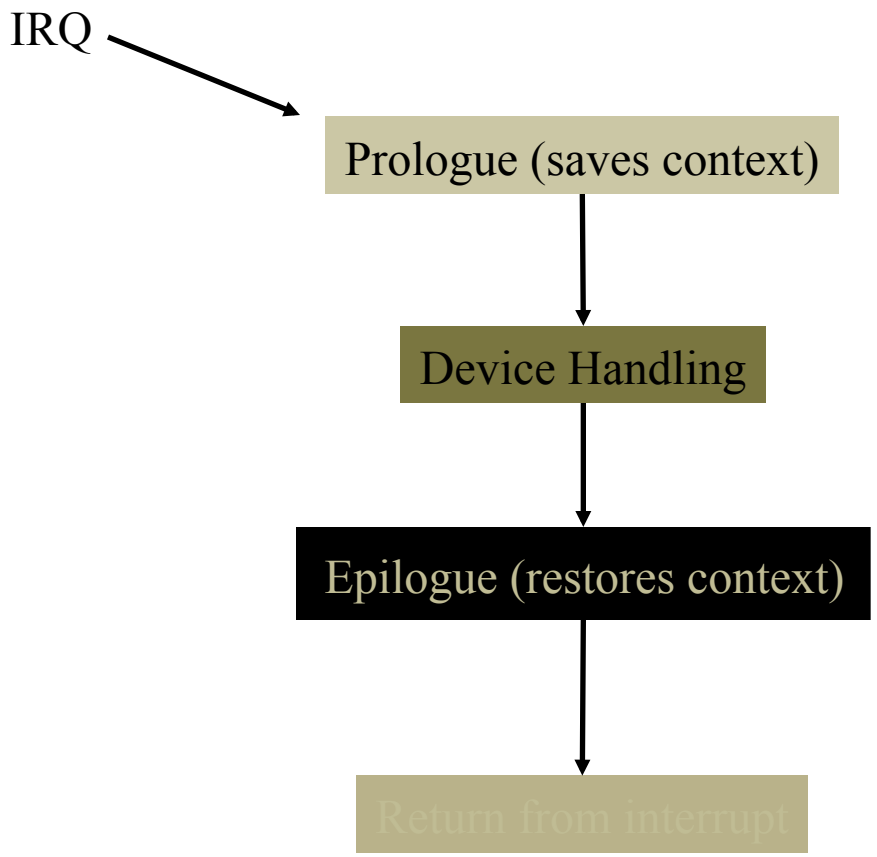
Interrupt Handling

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

INTERRUPT HANDLING ROUTINES

TRADITIONAL METHODOLOGY

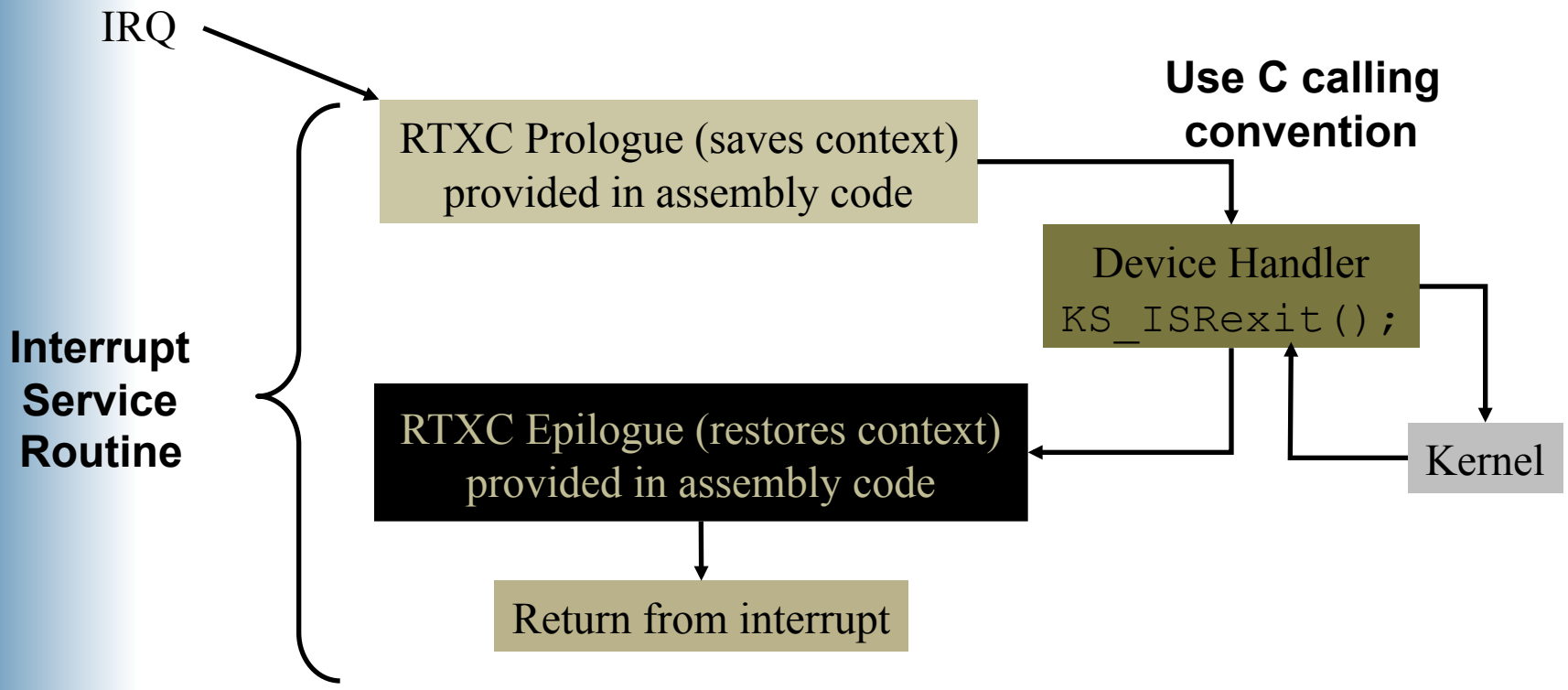


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC INTERRUPT HANDLING

RTXC METHODOLOGY

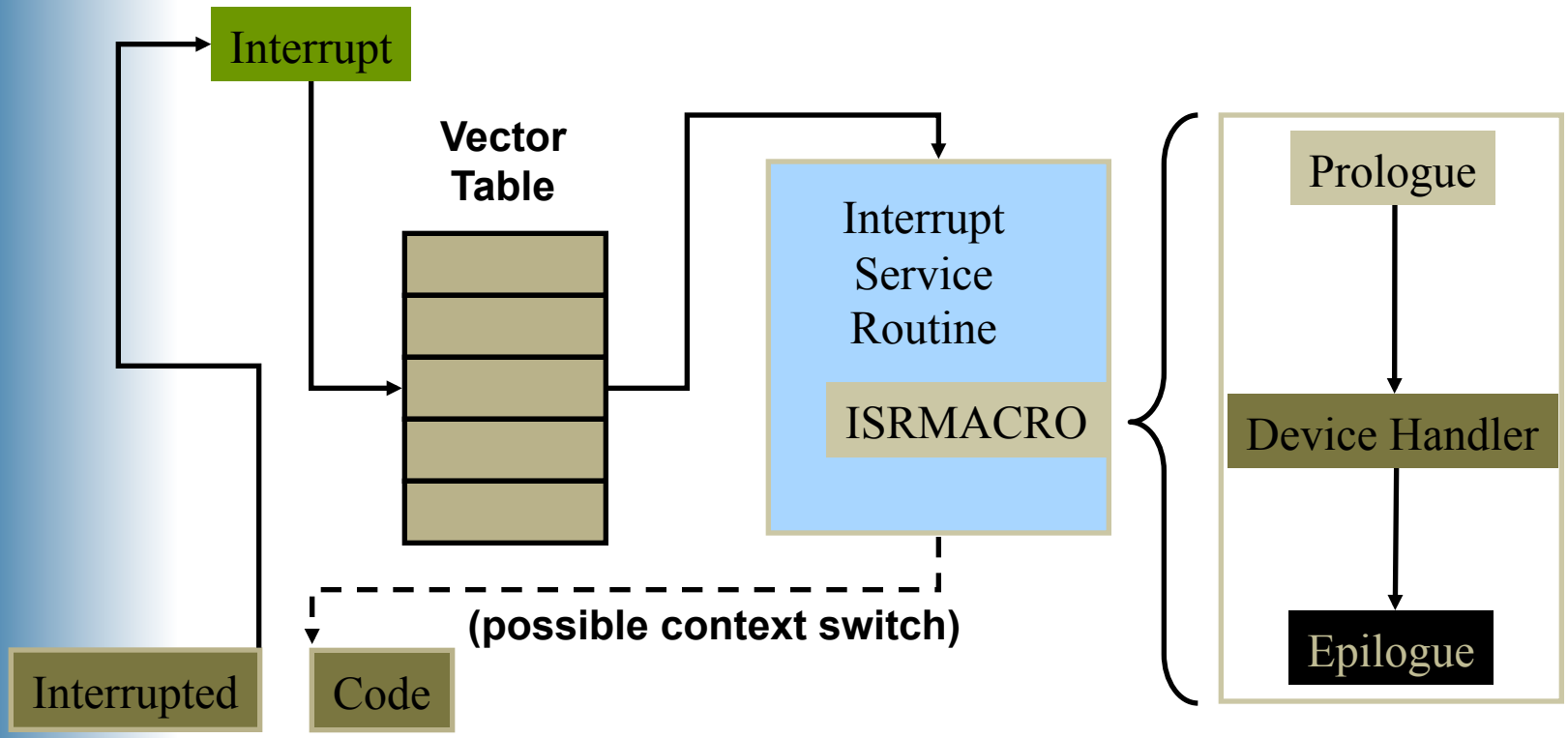


Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC INTERRUPT HANDLING

PROCESSING AN INTERRUPT



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC INTERRUPT HANDLING

RTXC PROLOGUE

- RTXC switches stacks from the task stack to the system stack, thus reducing the size of task stacks
- Saves interrupted context
- Interrupts are disabled while setting up contexts and processing interrupt nesting

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC INTERRUPT HANDLING

DEVICE HANDLER

- Written as a C function
  - Argument is pointer to the stack frame of the interrupted process
- Processes the interrupt
- Can be interrupted by higher level interrupts
- Only those kernel services denoted by KS_ISRxxx are allowed in the Device Handler
- Device Handler returns a frame pointer to the highest priority task so that the epilogue can restore that task's context

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

RTXC INTERRUPT HANDLING

INTERRUPT SERVICE ROUTINE SERVICES

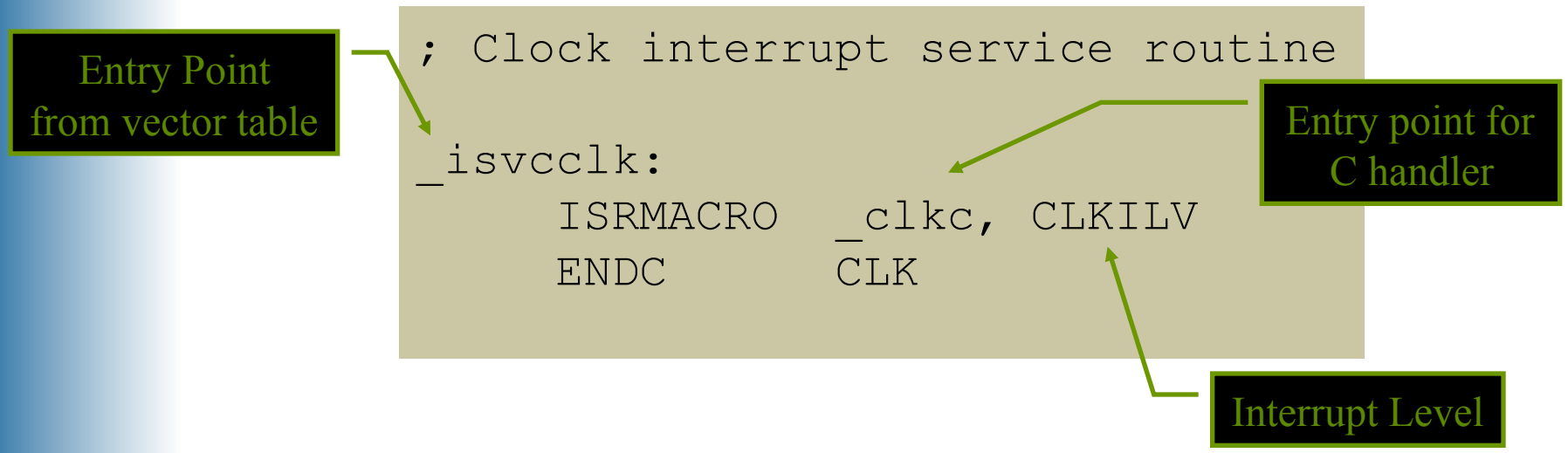
- `KS_ISRsignal(semaphore)`
  - Signal the given `semaphore` from an interrupt service routine
- `KS_ISRsignalm(semaphore_list)`
  - Signal all semaphores in the semaphore_list from the interrupt service routine
- `KS_ISRalloc(map)`
  - Allocate, from an interrupt service routine, a block of memory from `map` and return its address, returns NULL if `map` is empty
- `KS_ISRexit(frame,semaphore)`
  - Exit current interrupt service routine and optionally signal the given `semaphore`
- `KS_ISRtick(void)`
  - Perform RTXC required processing for a clock tick interrupt

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

ISRS.S

INTERRUPT SERVICE ROUTINE - ASSEMBLY CODE



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

CLKDRV.C

A SAMPLE C HANDLER

```
FRAME *clkc (FRAME *frame)
{
        KS_ISRtick();
        return(KS_ISRexit(frame, (SEMA)0));
}

/* clock driver start hook */
void clkstart(void)
{
        setvect(CLKINT,isvcclk);
        /* set interrupt level if needed */
```

Entry point for C handler

No semaphore to signal

Set the interrupt vector

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

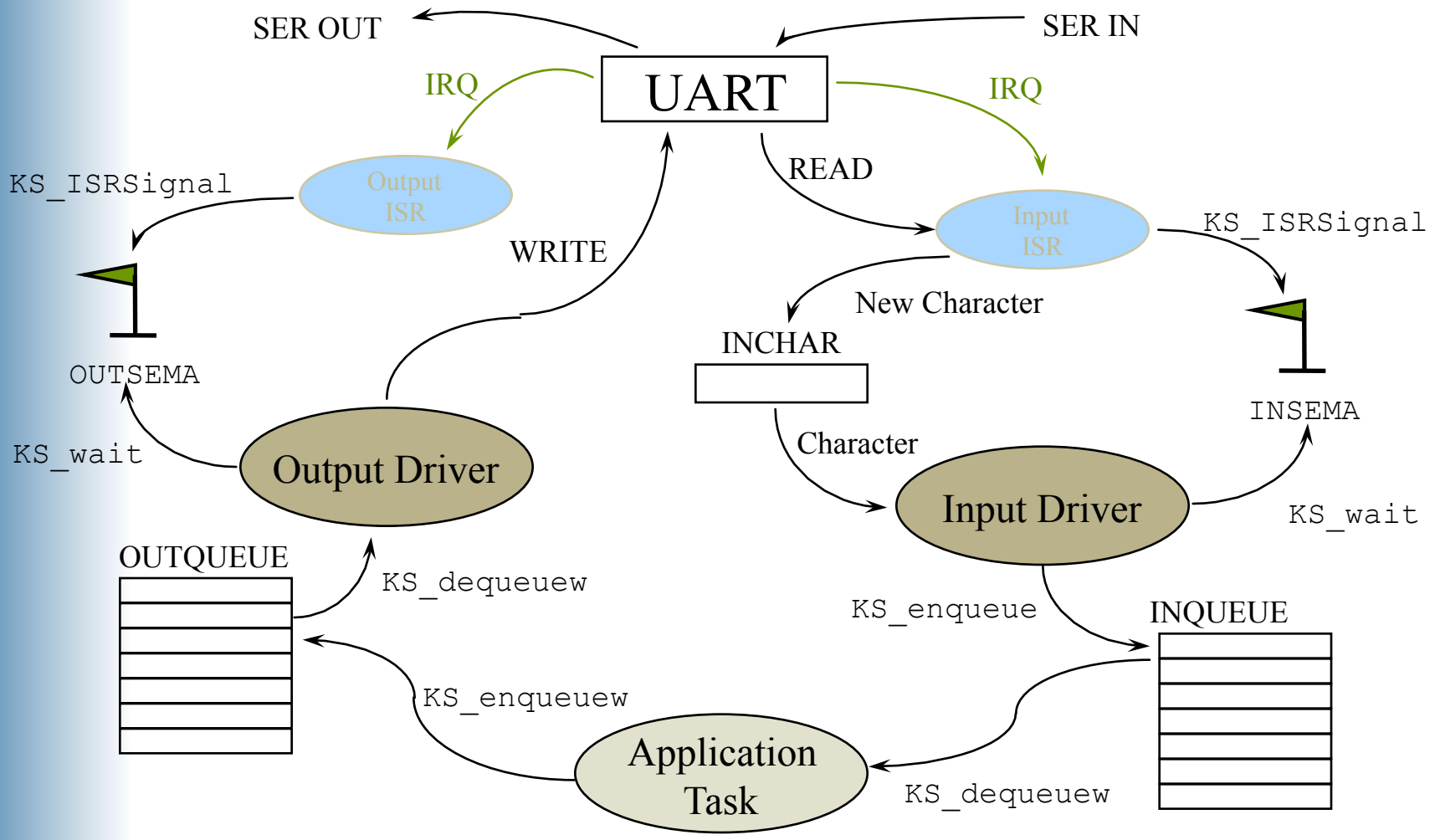
UART C HANDLER

```
FRAME *comc(FRAME *frame)
{  unsigned char status;
   status = inp(UART_INT_ID_REG);       /* read hw status */
   if (status == RX_DATA_READY) {       /* RX is higher priority w/in USART */
      ichar = inp(UART_RBR);            /* read char, clear USART interrupt */
      status = inp(UART_INT_ID_REG);    /* re-read hw status */
      if (status == TX_BUFF_EMPTY)
        KS_ISRsignal(OSEM);             /* signal char output semaphore */
      outp(PIC_EOI_ADDR, EOI);          /* clear PIC interrupt */
      return(KS_ISRexit(frame, ISEM));
   }

         /* signal char input semaphore */
   else {                                                                    /

   * A Transmit IRQ */
      if (status == TX_BUFF_EMPTY) {
        status = inp(UART_INT_ID_REG); /* re-read hw status */
        if (status == RX_DATA_READY)
          KS_ISRsignal(ISEM);            /* signal char input semaphore */
        outp(PIC_EOI_ADDR, EOI);         /* clear PIC interrupt */
        return(KS_ISRexit(frame, OSEM));/* exit signalling char output sema */
      }
   }
   outp(PIC_EOI_ADDR, EOI);              /* clear interrupt */
   return(KS_ISRexit(frame, (SEMA)0));
```

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.

SIMPLE DRIVER EXAMPLE



Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC INTERRUPT HANDLING

PRIVATE ISR

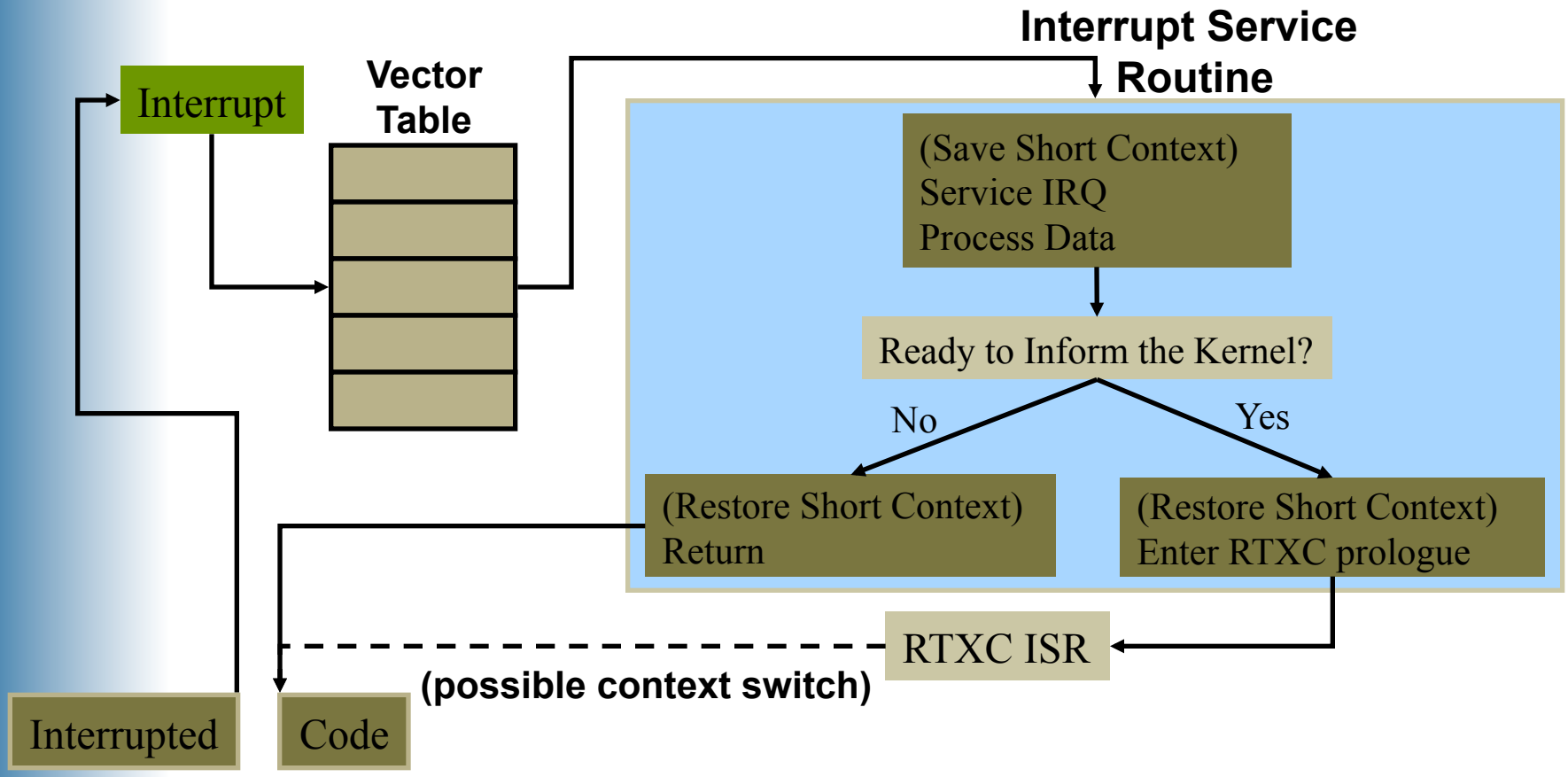
- The Interrupt Handler is outside the Kernel
- No Prologue – (can save a short context for increased performance)
- Can save stack switching logic if interrupts are disabled
- Service the IRQ
- (Restore short context) then decide what to do
  - Could just return from interrupt (fastest)
  - Or, if ISR runs at Kernel Ceiling Level, enter the regular RTXC ISR form starting with a prologue (use this to inform a kernel entity that an action needs to be taken)

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC INTERRUPT HANDLING

A PRIVATE INTERRUPT AT KCL*



*** Kernel Ceiling Level**

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros Systems Inc.

RTXC™

Real-Time Executive in C

Thank you!

Copyright © 2008, Quadros Systems, Inc., All Rights Reserved

Quadros™ Systems Inc.